The Thesis committee for Dongyi Zhou

Certifies that this is the approved version of the following thesis:

Hybrid Control and Model-based Assertions for Autonomous Intersection Management System as a Cyber-physical System

APPROVED BY

SUPERVISING COMMITTEE:

Raul G. Longoria, Supervisor

Benito R. Fernandez

Hybrid Control and Model-based Assertions for Autonomous Intersection Management System as a Cyber-physical System

by

Dongyi Zhou, B.E.

Thesis

Presented to the Faculty of the Graduate School
of the University of Texas at Austin
in Partial Fullfillment
of the Requirements
for the Degree of

Master of Science in Engineering

The University of Texas at Austin
May 2016

Acknowledgments

This research was supported by a National Science Foundation Grant Number CNS-1239498, titled "CPS: Synergy: Physically-Informed Assertions for CPS Development and Debugging.

Hybrid Control and Model-based Assertions for Autonomous Intersection Management System as a Cyber-physical System

by

Dongyi Zhou, MSE

The University of Texas at Austin, 2016

SUPERVISOR: Raul G. Longoria

A cyber-physical system (CPS) consists of multiple physical components that collaborate through a network during real-time operation according to system-level commands. A hybrid control, which generates discrete system-level commands and handles the low-level physical dynamics for each component, is of singular importance to a CPS. For a CPS, liveness and safety require that the system is always eventually doing what is desirable without any undesirable behavior, and have to be carefully addressed. The advance in information technology and autonomous driving make it possible to establish an autonomous intersection management system (AIMS) for ground autonomous vehicles, which can potentially improve traffic efficiency and reduce intersection car accidents. This work presents a hybrid control and introduces model-based assertions for such an AIMS. System-level traffic requirements are expressed in the form of linear temporal logic (LTL) specifications in the generalized reactivity formulas with rank 1 (GR(1)) and a two-player game is solved to synthesize a discrete traffic network controller. DaNI, a motor-driven laboratory robot vehicle, is modeled using a bond-graph approach, and a nonlinear vehicle trajectory sliding-mode controller

(SMC) is mathematically and numerically described. The discrete traffic controller instructs each robot vehicle to pass through the intersection of interest at a certain time and the vehicle's trajectory within the intersection is controlled by the SMC. Using simulation, it is shown that the synthesized discrete controller is able to determine proper system actions in response to traffic data and environmental actions while maintaining liveness and safety specifications. A laboratory study on a simple AIMS is demonstrated to depict the basic design structure of a AIMS as a CPS. The results suggest that model-based assertions, which monitor and validate a CPS by assertions based on physical models, can be helpful in detecting physics-related abnormalities during operation of a CPS that may not be captured by a software-level analysis. Using simulations, it is shown how the accuracy of a continuous vehicle trajectory controller, such as a SMC, can provide informative guidance on the design of model-based assertions.

Contents

List of Tables **x**

List of Figures **xi**

Chapter 1: Introduction **1**

Chapter 2: High-Level Discrete Control Synthesis for Traffic Network **6**

2.1 Traffic network model 8

2.2 FTS and two-player game for over-approximated traffic network 10

2.3 FTS and two-player game for concrete traffic network 13

2.4 Simulation and results 15

2.5 Integration of traffic discrete control into AIMS 18

Chapter 3: Implementation of AIMS with Model-Based Assertions **21**

3.1 AIMS in this laboratory research 22

3.1.1 The intersection 22

3.1.2 The ground robot vehicle 23

3.1.3 The intersection network 25

3.1.4 Sensors on DaNI 25

3.2 Program architecture 27

3.2.1 LabVIEW programming environment 30

3.2.2 Initialization . . . 31
3.2.3 Loop on the intersection manager . . . 31
3.2.4 Low-priority loop on DaNI . . . 32
3.2.5 High-priority timed loop on DaNI . . . 33
3.3 Assertions and tests . . . 36
3.3.1 Initialization check . . . 37
3.3.2 Ambient light check . . . 38
3.3.3 DaNI stop check . . . 38
3.3.4 DaNI not responding . . . 39
3.3.5 Undesired motions check . . . 40
3.4 AIMS: laboratory study to practical implementation . . . 41

Chapter 4: Robust Trajectory Control for DaNI **43**

4.1 Model . . . 45
4.2 SMC trajectory controller . . . 49
4.2.1 Controllability and observability . . . 49
4.2.2 Design of sliding-mode controller . . . 50
4.2.3 Pre-filter . . . 54
4.2.4 Extended luenberger observer . . . 54
4.3 Simulation of SMC controller . . . 55
4.3.1 Simulation workflow . . . 56
4.3.2 Simulation results . . . 58
4.4 Impact of continuous control on model-based assertions . . . 63

Chapter 5: Conclusion **65**

Appendix A: Code for the laboratory AIMS **67**

Appendix B: Code for continuous trajectory control **70**

Appendix C: Code for discrete traffic control **83**

Bibliography **89**

List of Tables

2.1 Sample traffic network parameters 16

3.1 Test results for validation of vehicle motion assertion 41

4.1 Model parameters for the 2-D dynamic model of DaNI 48

List of Figures

1.1 An autonomous intersection with a local manager and autonomous vehicles (A, B). The vehicles communicate to the manager their operational data (speed, motion plan, etc). The manager transmits back commands. 3

2.1 Memory requirements in MATLAB 14

2.2 Sample traffic network 15

2.3 Synthesized controller by TuLiP for over-approximated traffic network 17

2.4 Synthesized controller by TuLiP for concrete traffic network 18

2.5 Concrete traffic state evolution 19

3.1 Intersection in this laboratory research 23

3.2 Operation block diagram of DaNI (ni.com) 24

3.3 VN-100 inertial measurement unit (vectornav.com) 26

3.4 Incremental encoder kit on DaNI (pitsco.com) 26

3.5 Optical line sensor (vexrobotics.com) 27

3.6 Program architecture of the AIMS using a FCFS reservation protocol 29

3.7 Finite state machine for intersection reservation assignment 32

3.8 Line sensor layout upon entrance to and departure from the intersection . . . 33

3.9 Body-fixed reference frame for DaNI 35

3.10 Possible trajectories of a DaNI within an intersection 35

4.1 Schematic for a typical SMC feedback control system ([23]) 44

4.2 2-D bond graph without slip for DaNI. SMC stands for sliding-mode controller; ELO stands for extended Luenberger observer; Measurements includes path length and yaw angle. 46
4.3 Simulation with no modulation resolution restriction, observer, error, or noise 60
4.4 Simulation with modulation resolution restriction, but without observer, error, or noise 61
4.5 Simulation with modulation resolution restriction, observer, error, and noise . 62
6.1 Intersection Manager LabVIEW front panel and block diagram 68
6.2 DaNI LabVIEW front panel and block diagram 69

Chapter 1: Introduction

Consider a cyber-physical system (CPS), where each component works individually in its physical domain while collaborating with other components and a system manager through a network. A cyber-physical system interacts with the physical environment bidirectionally and, as a result, its performance can be intensely influenced by the physics it exhibits. To build a reliable cyber-physical system, synthesis and distribution of commands must be carefully established to assure system-level functionality. In addition, proper low-level control of each individual component is required to ensure system-level collaborations.

During real-time operation, safety and liveness are recognized as the two most critical requirements to qualify a cyber-physical system. Safety requires the operation of a cyber-physical system poses nothing dangerous to itself and the environment. Liveness defines the ability of a cyber-physical system to complete assigned missions without failure at any level. To address liveness and safety, a cyber-physical system has to be monitored, controlled, and validated throughout real-time operation. This research specifically studies hybrid control and model-based assertions for an autonomous intersection management system (AIMS) for autonomous ground vehicles, which inherently forms a cyber-physical system. Model-based assertions are logical statements based on interpretation of models, such as a traffic or a vehicle dynamics model, and measurements data, such as the vehicle yaw angle. Recent advances in autonomous vehicles reduces the response time of vehicle driving and promises improvement to traffic efficiency [1]. The promising future and the popularity of autonomous

ground vehicles will tremendously improve the driving efficiency [2]. An intersection management system (IMS) is a type of vehicle-to-infrastructure system that can provide support to vehicles in relation to the roadway infrastructure. An AIMS automatically generates discrete commands based on current traffic data to meet system-level specifications, and can be essential to exploit the superior efficiency of autonomous vehicles [2]. In addition, a National Motor Vehicle Crash Causation Survey reveals that about 36 percent of crashes take place at intersections and therefore research on intersection management is of great significance to accident reduction [3]. While many studies have been focusing on either autonomous vehicle control or autonomous intersection management, this thesis presents a preliminary study on AIMS for autonomous vehicles as a cyber-physical system regulated by a hybrid control, and a model-based assertion approach to monitor and verify the safety and liveness of the controlled AIMS.

Consider a basic four-way intersection to which self-driving vehicles approach in Figure 1.1. Instead of traffic lights, this intersection is managed by a management computer that distributes commands to each approaching vehicle. Traffic commands are synthesized systematically at the level of a traffic network, which will be discussed in detail later in this thesis. All local intersection managers together form up the computational unit of the traffic network. In an AIMS as a CPS, liveness requires that all vehicles finally pass through the intersection while safety requires that they pass without collision. A DaNI programmable mobile robot vehicle (National Insturments, Austin, TX), was used to implement test methods and ideas developed in this thesis research.

A hybrid control for AIMS has to deal with both traffic dynamics and vehicle dynamics. To begin with, a discrete planner synthesizes high-level traffic activation commands that meet traffic network specifications. Then, a continuous controller follows the discrete commands and handles low-level vehicle dyanmics. In this approach, vehicle motion is controlled by a sliding-mode vehicle trajectory controller designed upon a 2-D vehicle dynamics

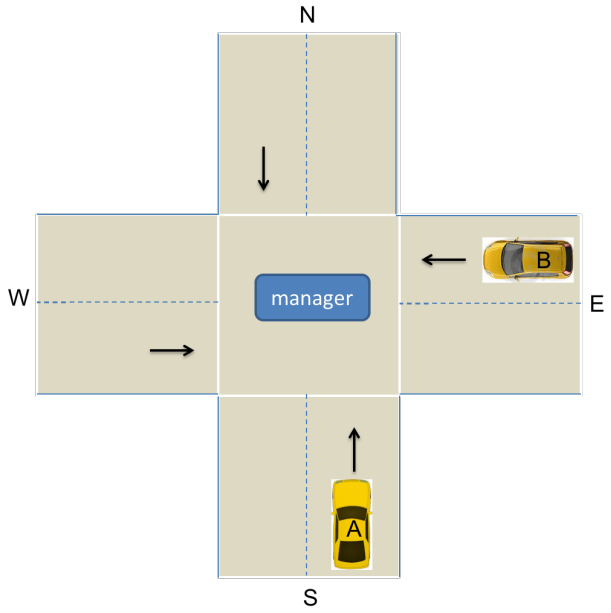

Figure 1.1: An autonomous intersection with a local manager and autonomous vehicles (A, B). The vehicles communicate to the manager their operational data (speed, motion plan, etc). The manager transmits back commands.

model. Given a desirable trajectory in time domain, sliding mode control (SMC) is able to regulate the motion so that the vehicle follows a desirable trajectory even in the presence of model error, measurement error, plant disturbance, and control input constraints.

Traffic flow control is the subject of many research studies. Stage-based strategies such as SIGSET, provide fixed-time traffic control to minimize the total delay [4]. MAXBAND is a fixed-time coordinated traffic control that maximizes the number of vehicles that can continue traveling through an intersection without any stop [5]. SCOOT utilizes real measurements, such as traffic capacity and volume, to repeatedly make incremental changes to traffic control parameters [6]. In this research, linear temporal logic (LTL) is applied to represent traffic network specifications in a proposed traffic network control synthesis. A concrete (original) traffic network can be abstracted into an over-approximated traffic network (defined in Chapter 2) that is able to suggest an infeasibility if the traffic network

specifications are too restrictive to be realized. An over-approximated traffic network has much fewer states than the actual network and is thus computationally more efficient. It could provide preliminary guidance on the design of traffic network properties and specifications. Once verified in the over-approximated traffic network, specifications can be loaded to the concrete system to synthesize a discrete traffic flow control strategy using the Temporal Logic Planning (TuLiP) toolbox [7]. TuLiP is a Python-based software toolbox for control synthesis of finite state abstraction of systems based on linear temporal logic (LTL) specifications in the generalized reactivity formulas with rank 1 (GR(1)) [7, 8].

Due to the inherent complexity of CPS and the intense interaction between multiple domains, compositional "verification and validation" is necessary [9]. Conventional solutions such as overdesign become inadequate or impractical for CPS and thus a new methodology is needed [9]. Assertions are widely employed in software engineering to examine correctness in code. For example, an assertion is used to check whether a divisor is nonzero when calculating remainder [10]. This research also seeks to advance the formulation and application of model-based assertions on cyber-physical systems so as to examine both logical and physical status [11].

An assertion is most commonly in the form of a logical statement based on a software model. A physical model enables us to assert about a cyber-physical system if the model provides a good representation of the system physics [12]. For example, for cyber-physical systems where system components may interact with human beings, model-based assertions can give us confidence in predicting and preventing potential threats. Apart from assertions that are solely logic, cyber-physical systems should also involve assertions based on physical models that encode system components and their interaction with the environment and each other. For example, ambient light density should be assessed if a CPS utilizes a light-based sensor. In such an assertion involving physics, physics-based models can provide expected status of the CPS compared to the actual system status. Various sensors customized to

a specific application should be integrated into the CPS to provide system status using measurement data. Conventionally, a cyber-physical system like an AIMS will be tested many times to validate a certain level of confidence in performance. It is hypothesized that model-based assertions, if properly designed and distributed into the system, may provide a systematic and consistent way to debug and respond to unexpected behavior of the system during operation. To achieve this goal, assertions need to be developed both for preliminary testing and for real-time operation.

In order for an AIMS manager to assign a right of way, each approaching vehicle has to share its state and proposed route with the intersection manager and receive commands from the manager. Therefore performance of the intersection manager is subject to reliable communication with vehicles. As for the vehicles, they must follow commands from the manager and conduct a proposed motion plan as scheduled. Their performance is subject to communication, sensing, actuation, and influence of the physical environment. Besides verification of a valid request from the manager, assertions are placed within the vehicle to check motions. For example, vehicle speeds are checked after being triggered to stop. After being allowed to proceed into the intersection, each autonomous vehicle needs to check whether it is passing through the intersection as planned. For example, has the vehicle moved onto the opposite road entrance if it was planning to go straight? In the case of any assertion failure, an alarm can be displayed to locate the error, and corresponding action such as an emergency stop can be taken.

In this thesis, the high-level discrete control synthesis for traffic network is presented in Chapter 2. Then the implemention of AIMS with robot vehicles and model-based assertions in AIMS is discussed in detail in Chapter 3. Next, a sliding-mode vehicle trajectory controller for low-level vehicle control is described in Chapter 4. Finally, the thesis summarizes the work with conclusions in Chapter 5.

Chapter 2: High-Level Discrete Control Synthesis for Traffic Network

The discrete controller handling the traffic dynamics in an intersection management should be able to instruct each vehicle to pass through the intersection or not at each time step. Briefly speaking, it generates the high-level commands for each vehicle that guide the low-level continuous vehicle controller. For a traffic network control, the system action is the activation of certain roads, and the environmental action is the planned motion for each vehicle, i.e., going straight, turning left, or turning right, and the extraneous traffic flow. The control is to decide on roads to be activated in response to real-time traffic such that the entire traffic network meets liveness and safety criteria as well as traffic design specifications. To establish the high-level discrete control for an AIMS, a traffic network model from [13] is adopted with slight modifications. Finite transition systems (FTS) for the traffic network are generated according to the evolution dynamics regarding the number of vehicles on each road. A control strategy based on the FTS is synthesized by solving a two-player game between environmental action and system action. Linear Temporal Logic (LTL) is applied in this work to describe system temporal specifications [14].

Any LTL specification is composed of basic syntax,

$$\varphi(\text{atomic proposition}),\ \varphi_1 \wedge \varphi_2,\ \neg\varphi,\ \bigcirc\varphi(\text{next}),\ \varphi_1 U \varphi_2(\text{until}), \tag{2.1}$$

and, derived syntax,

$$\Box\varphi(\text{always}),\ \Diamond\varphi(\text{eventually}),\ \varphi_1 \to \varphi_2(\text{imply}). \tag{2.2}$$

An example of LTL specifications for a traffic network can be "the number of vehicles on a road (x) is always eventually less than N", or,

$$\Box\Diamond(x < N). \tag{2.3}$$

Here, $x < N$ is an atomic proposition. A particular subclass of LTL specifications is the GR(1) specification. A GR(1) specification can be uniformly expressed as,

$$\begin{aligned}&(env_{init}\ \&\ \Box env_{safety}\ \&\ \Box\Diamond env_{prog1}\ \&\ \Box\Diamond env_{prog2}\ \&\ \cdots)\\ \to\ &(sys_{init}\ \&\ \Box sys_{safety}\ \&\ \Box\Diamond sys_{prog1}\ \&\ \Box\Diamond sys_{prog2}\ \&\ \cdots).\end{aligned} \tag{2.4}$$

$\Box$ means a proposition always holds and addresses the safety requirement of a CPS. $\Box\Diamond$ means a proposition always eventually holds for infinite times and ensures the liveness of a CPS. In addition, it has been shown that control synthesis for specifications in the GR(1) form can dramatically reduce the computational complexity [8].

An unsolved two-player game can contain a great many of transitions under different environmental actions and system actions. Some transitions, though realizable according to traffic dynamics, may violate certain LTL specifications. TuLiP is a control synthesis tool that can deal with a two-player game of this kind with the GR(1) specification [7]. It is able to decide on which control to take in order to satisfy the specifications.

A traffic network can be analyzed as a concrete system or an abstracted system. In an abstracted system, an abstracted state can contain a number of concrete states. For an abstracted state, if there exists a concrete state within it that can transition into a concrete state in another abstracted state, then we say an abstracted transition exists between these two abstracted states. Abstracted states, along with all feasible abstracted transitions, form up an over-approximation of the traffic network. To generate an over-approximated traffic net-

work that abstracts the traffic dynamics, a "proposition-preserving partitioning" is necessary. That is to say, if one concrete state in an abstracted state satisfies an atomic proposition, all other concrete states in that abstracted state must also satisfy the same proposition. If there is no feasible transition from an abstracted state in the over-approximation that meets the specifications, no concrete state within that abstracted state can transition into any other concrete state while maintaining the specifications. Thus, if a controller is not realizable for the over-approximated traffic network, there won't be any control for the concrete traffic network that can meet the specifications. In this work, TuLiP first tries to synthesize a controller for the over-approximated traffic network. If not realizable, then we may go back to adjust traffic network parameters or specifications. If a control is successfully synthesized, then we may start synthesizing a controller for the concrete traffic network with the same specifications. As the over-approximated system contains far less number of states, a lower computational cost is needed to suggest an infeasibility of the specifications if they are indeed unrealizable.

This chapter first introduces the slightly modified traffic network model, then the finite transition systems and two-player game for both over-approximated and concrete traffic networks are generated. At last, this chapter concludes with simulated traffic control synthesis for a simple traffic network.

2.1 Traffic network model

A traffic network model consists of roads(links), intersections(nodes), and vehicles. In order for segmented traffic networks to form up a single global traffic network, the model must start with only links and end with only nodes (or start with only nodes and end with only links). From now on, let's use $L = \{l_i\}$ to denote all links i, $V = \{v_j\}$ to denote all nodes, and $\underline{x} = [x_i]$ to encode the numbers of vehicles on l_i at each time step. Every link has a maximum number of vehicles it can hold at one time and this is represented by $x^{cap} = [x_i^{cap}]$.

Also, as the time step decreases, the maximum allowable outflow for each link decreases. Outflow threshold is denoted by $c = [c_i]$.

For each node v_i, a control strategy may activate a partial set of links connected to that node at each time step, $s_v = \{l_i@v\}$. If $S_v = \{s_v\}$ denotes all possible activated sets of links for node v, then a system action for the traffic network will be a set of all links that can be activated at the same time, expressed,

$$S = \{s = \bigcup_{v \in V} s_v \mid s_v \in S_v\} \subseteq 2^L. \tag{2.5}$$

To fully depict the traffic dynamics, two more variables have to be defined: turn ratio and supply ratio. Turn ratio stands for the percentages of vehicles on a link that propose to go straight, turn right, turn left, or make a U-turn. Supply ratio determines percentage of space left on a link that is available to a certain upstream link. Specifically,

$$\begin{aligned} \beta_{lk} &- \text{turn ratio: fraction of vehicles exiting link } l \text{ that are routed to link } k \\ \alpha_{lk} &- \text{supply ratio: fraction of link } k\text{'s capacity available to link } l. \end{aligned} \tag{2.6}$$

The traffic dynamics discussed in [13] do not take into account the fact that x can only be non-negative integers. Here, we ceil the x calculated following the traffic dynamics in [13]. In this way, it is ensured that the traffic states only evolve to non-negative integers. In addition, taking the *ceil* is safe because we can always assume there is one more vehicle on each link. This approach will virtually reduce the capacity by 1 for some scenarios but will keep the traffic states as non-negative integers. The modified traffic dynamics has two parts: outflow for each link, and number of vehicles on each link at next time step. If a link is activated for the current time step, its outflow is the minimum among current number of vehicles on it, its outflow threshold, and maximum outflow based on available downstream spaces. If a link is not activated, then its outflow is 0. The number of vehicles on a link at the next time step is the minimum between its capacity and the number accounting for

extraneous input flow, outflow, and inflow from upstream links, or,

$$x_l(t+1) = \left\lceil \min\{x_l^{cap}, x_l(t) - f_l^{out} + d_l(t) + \sum_{j \in l_l^{upstream}} \beta_{jl} f_j^{out}\} \right\rceil$$
$$f_l^{out}(x(t), S) = \begin{cases} \min\{x_l(t), c_l, \min\{\frac{\alpha_{lk}}{\beta_{lk}}(x_k^{cap} - x_k(t))\}\} & \text{if } l \in S \\ 0 \quad \text{otherwise} \end{cases} . \tag{2.7}$$

This equation defines the update principle for number of vehicles on each link, and is of great significance to the formation of the transition system.

2.2 FTS and two-player game for over-approximated traffic network

To establish a finite transition system for the over-approximated traffic network, a "proposition-preserving partitioning", as discussed at the beginning of Chapter 2, is required. It can be anticipated that traffic network propositions are about number of vehicles. As a result, a good proposition-preserving partioning could be based on ranges of number of vehicles. Mathematically, for each link l_i, the j^{th} partition is $p_{l_i}^j = [m, n], m, n \in N$, and,

$$[0, x_i^{cap}] = \bigcup p_{l_i}^j \ \& \ \bigcap p_{l_i}^j = 0. \tag{2.8}$$

We can then form up the abstracted states in the over-approximated traffic network. Each abstracted state represents a set of ranges of number of vehicles on all links,

$$Y_{abstract}^j = \{p_{l_i}^j\}. \tag{2.9}$$

In this research, each link is divided into two segments. The lower segment indicates that the number of vehicles on that link is in the low range, while the other segment indicates a capacity pressure. If we don't care about the number of vehicles on a link, then that link is not divided and any value within the link capacity will be acceptable.

Next, a FTS can be computed for the over-approximated traffic network following the traffic dynamics. As proved in [13], the number of vehicles on a link increases with number of vehicles on downstream and upstream links, and extraneous traffic flow d_i, but decreases with number of vehicles on adjacent links. Adjacent links are those that share the same upstream node with l_i. With this componentwise monotonicity property, the reachable set for an abstracted state can be evaluated. For example, assume the abstracted system is at a certain state $Y^i_{abstract}$, where the number of vehicles on each link and extraneous input flow all lie in a bounded range $[m, n]$. Obviously bounded ranges will yield bounded ranges for all links. To evaluate the upper bound of the number of vehicles on l_k, we use the upper bounds for upstream, downstream links of l_k and input flow, but the lower bounds for adjacent links of l_k. Then according to the monotonicity, this calculation will generate the maximum possible value for link l_k if evolved from abstracted state $Y^i_{abstract}$. The expected lower bound for link l_k can be calculated in a similar way. Repeating this calculation for all links, a reachable set for $Y^i_{abstract}$ is obtained. Next, if the reachable set intersects with an abstracted state $Y^j_{abstract}$ in each dimension (for each link), there is a transition from $Y^i_{abstract}$ to $Y^j_{abstract}$. Transitions for the over-approximated traffic network is computed in MATLAB, as shown in the algorithm below.

```
function suc = successors(ylow, yhigh,Qpar,Qname)
% ylow and yhigh are the lower and upper bound of the reachable set
% Qpar encodes the partions of the network
% Qname contains the name of abstracted states
suc = {};
for i = 1:length(Qname)
    if all(Qpar(1,:,i) <= yhigh)  && all(Qpar(2,:,i) >= ylow)
        suc{length(suc)+1} = Qname{i};
    end
```

```
end
```

In a two-player game, the system wins if it always selects such system actions that specifications are satisfied, in response to any environmental action under environmental assumptions. System action for an over-approximated traffic network is a set of activated nodes, $s \in S$. Extraneous traffic flow is modeled as a bounded set and incorporated into the FTS construction. As a result, it will not show up explicitly as an environmental variable here. The only environmental variable is then the "turn ratio". "Turn ratio" is collaboratively determined by all the drivers of vehicles within the traffic network of interest, and is thus not controllable by the traffic managers. For example, if 3 out of 10 vehicles on link l_i plan to make a left turn to link l_j, then $\beta_{ij} = 0.3$. Theoretically speaking, "turn ratio" can assume any value such that $(x_i^{cap} \cdot \beta_{ij})$ is a non-negative integer. For a research purpose, two distinct sets of "turn ratios" are used to represent variation in "turn ratio".

In this particular two-player game constructed on an abstracted network, the global traffic control planner has to select the right s so that all transitions satisfy the GR(1) specification no matter the "turn ratios". A separate transition matrix is computed for each combination of s and a set of "turn ratios". If TuLiP fails to synthesize a controller, it means at some abstracted state and under a certain set of "turn ratios" there does not exist a set of activated nodes and an extraneous input flow in range, such that the traffic network can transition into a valid state while satisfying the specifications. Further, it means no concrete state contained in that abstracted state could transition into any other concrete state while maintaining specifications. If so, there is no need to try to synthesize a controller for the concrete system with the same specifications.

The scalability of the problem depends on the number of nodes, the number of links connected to each node, and the number of atomic propositions involved in the GR(1) specifications. However, if we add nodes or links that will not be involved in any specification, then abstracted states will not change. The number of transitions in the synthesized control

may vary a little due to additional dynamics. Whenever a newly added link is involved in the GR(1) specifications, the number of over-approximated states will double.

How well an over-approximation predicts the realizability of a GR(1) specification is essential to this approach. As discussed earlier, if a GR(1) is not realizable in the over-approximated traffic network, it is by no means realizable in the concrete traffic network. Section 2.4 will provide more details regarding the ability of over-approximation to predict infeasibility of GR(1) specification .

2.3 FTS and two-player game for concrete traffic network

Each concrete state contains a specific number of vehicles for each link, $Y_{concrete} = [x_i]$. It is much easiler to construct the transitions for a concrete traffic network. Actually, for each concrete state, there is one and only one state it can transition into for specific "turn ratios" and fixed extraneous input flow. An algorithm to calculate a concrete system transition matrix is shown below.

```
% Signal contains all feasible sets of activated nodes
% States_Mat includes all concrete states
TS = false(N,N,length(Signal));
yindex = uint16(zeros(1,N));
for i = uint16(1:length(Signal))
    SignalTem = Signal{i};
    parfor j = uint16(1:N)
       y = uint8(trafficdyn(Xcap, Cflow, States_Mat(j,:), SignalTem, para, Dis));
       [~,yindex(j)] = ismember(y, States_Mat, rows);
    end
      TS(sub2ind(size(TS),1:N, yindex, uint16(ones(1,N))*i)) = 1;
```

```
end
```

As seen in the algorithm, TS is a 3-dimensional matrix that encodes the transitions between states for all possible set of activated nodes $s \in S$. The size of TS is NumofStates $\times$ NumofStates $\times$ size(S). It can be expected that the size of the concrete state space increases very quickly as x^{cap} or number of links increases. In particular, the size of concrete state set is $N_{state} = \prod_{l_i \in L} (x_i^{cap} + 1)$. For example, for a traffic network with 7 links and $x^{cap} = [3, 5, 5, 3, 3, 3, 3]$, $N_{state} = 36864$. As N_{state} becomes rather large, attention needs to be paid to the data type of variables. For example, initializing a transition matrix with a size of $36864 \times 36864 \times 4$ using *double* data type on a 8-GB-memory Mac will almost use up the memory. Thus careful memory preallocation is mandatory here. Instead of using a *double* data type for TS, a *logical* data type is able to reduce the size of TS to 1/8. Other variables also uses as simple a data type as possible. Figure 2.1 shows a comparison of bits requirements of different data types in MATLAB.

Class (Data Type)	Bytes
single	4
double	8
logical	1
int8, uint8	1
int16, uint16	2
int32, uint32	4
int64, int64	8

Figure 2.1: Memory requirements in MATLAB

For a two-player game upon a concrete traffic network, system action is still a set of simutaneously activated nodes. For environmental actions, besides "turn ratios", the lower and upper bound of extraneous input flow (D) in an over-approximated system are also taken as two environmental actions. That is, two distinct sets of "turn ratios" and

two distinct sets of input flows compose all possible environmental actions for this concrete traffic network. For each combination of input flow and "turn ratios" and each system action $s \in S$, there is a separate transition matrix. In a synthesized control, the system has to take a certain action in response to any eligible environmental action, such that the concrete state transitions always satisfy the GR(1) specification. This concrete state control can be implemented directly onto the traffic system.

2.4 Simulation and results

The sample traffic network simulated for this study is shown in Figure 2.2, with traffic network parameters summarized in Table 2.1. As discussed above, the traffic network starts with only links and ends with only nodes. Links $l_1, l_2, l_3, l_4, l_5, l_6$ and nodes v_1, v_2, v_3 are relevant to this study. s_{123} means links l_1, l_2, and l_3 are activated at the same time.

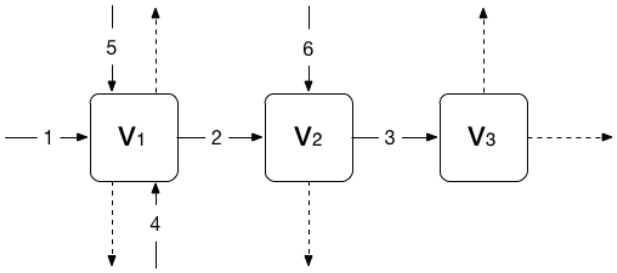

Figure 2.2: Sample traffic network

Now we create the GR(1) specification. For a concrete system, high input flow is true for infinitely many times and once input flow is high it is low in the next time step. Also, side links l_4, l_5 and l_6 are always eventually activated. And most importantly, the number of vehicles on main links l_1, l_2, l_3 and on link l_6 must eventually be always no more than corresponding $x_{l_i}^{low}$ to allow more extraneous input flow. In GR(1) form, they can be denoted as:

Table 2.1: Sample traffic network parameters

Parameter	Value
x^{cap}	[3, 5, 5, 3, 3, 3]
x^{low}	[2, 3, 3, 3, 3, 2]
c	[2, 3, 3, 1, 1, 2]
α (52,42,12, 23,63)	[0.5, 0.5, 1, 1, 1]
β_1 (52,42,12,23,63)	[0.5, 0.5, 0.5, 0.5, 0.9]
β_2 (52,42,12,23,63)	[0.2, 0.2, 0.4, 0.8, 0.2]
d_{low}	[0, 0, 0, 1, 1, 0]
d_{high}	[1, 0, 0, 2, 2, 1]
S	{s123, s163, s4523, s4563}

```
env_prog = {env_actions = turn1_Dishigh, env_actions = turn2_Dishigh}
env_safe = {((env_actions = turn1_Dishigh) || (env_actions = turn2_Dishigh)) ->
X ((env_actions = turn1_Dislow)  || (env_actions = turn2_Dislow)) }
sys_prog = {(sys_actions = s4523)  || (sys_actions = s4563)}
sys_prog |= {(sys_actions = s163) || (sys_actions = s4563)}
sys_prog |= {l1_low, l2_low, l3_low, l6_low}
sys_safe = {l1_low -> X (l1_low), l2_low -> X (l2_low), l3_low -> X (l3_low),
l6_low -> X (l6_low)}
```

"safe" means always satisfied and "prog" means always eventually satisfied.

For over-approximated traffic network, there are $2^4 = 16$ states in the form of $xabcd$, where a, b, c, d are 1 or 2 with 2 representing high vehicle number and 1 for low vehicle number. For instance, $x2211$ means the numbers of vehicles on l_1, l_2 are high, and numbers of vehicles on l_3, l_6 are low. In simulation, the initial state is set to $x2222$, which is the most adversarial case for a control synthesis. It takes TuLiP about 10 seconds to syntheisze a controller for the over-approximated traffic network. Figure 2.3 shows the resulting controller.

For concrete traffic network, there are 9216 states in total. Environmental actions are taken from the following set,

```
{turn1_Dislow, turn2_Dislow, turn1_Dishigh, turn2_Dishigh}
```

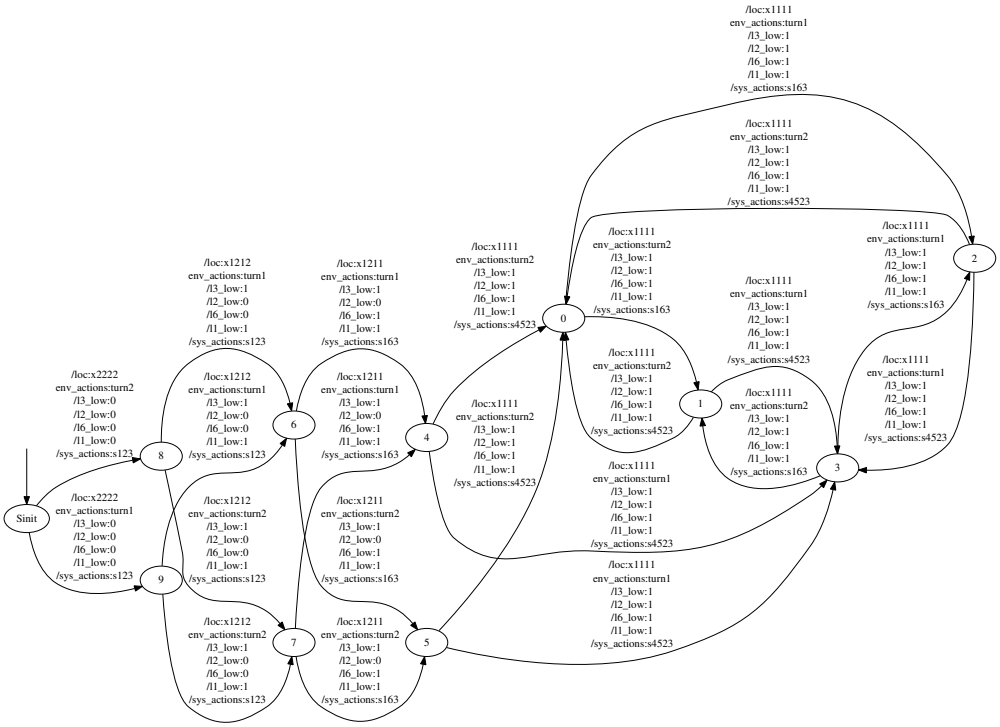

Figure 2.3: Synthesized controller by TuLiP for over-approximated traffic network

"turn2_Dishigh" means the second set of "turn ratios" is applied and extraneous input flow is high. It takes TuLiP about 20 minutes to synthesize a controller for the concrete traffic network. Figure 2.4 shows the synthesized controller. The evolution of number of vehicles is shown in Figure 2.5. As shown, number of vehicles on l_1, l_2, l_3, l_6 are eventually always within their low range.

Now, instead of requiring that the number of vehicles on link 1, 2, 3, and 6 are eventually always below [2,3,3,2], a more restrictive requirement of zero vehicles on link l_3 will make the GR(1) specification not realizable. This basically says all vehicles on link l_3 will be routed to downstream links and no vehicle will be routed to link l_3. A quick analysis of the traffic network will make it obvious that this GR(1) specification is not realizable. In simulation, TuLiP states that this modified GR(1) is not realizable in the over-approximated traffic network. It is expected that the same GR(1) will not be realizable

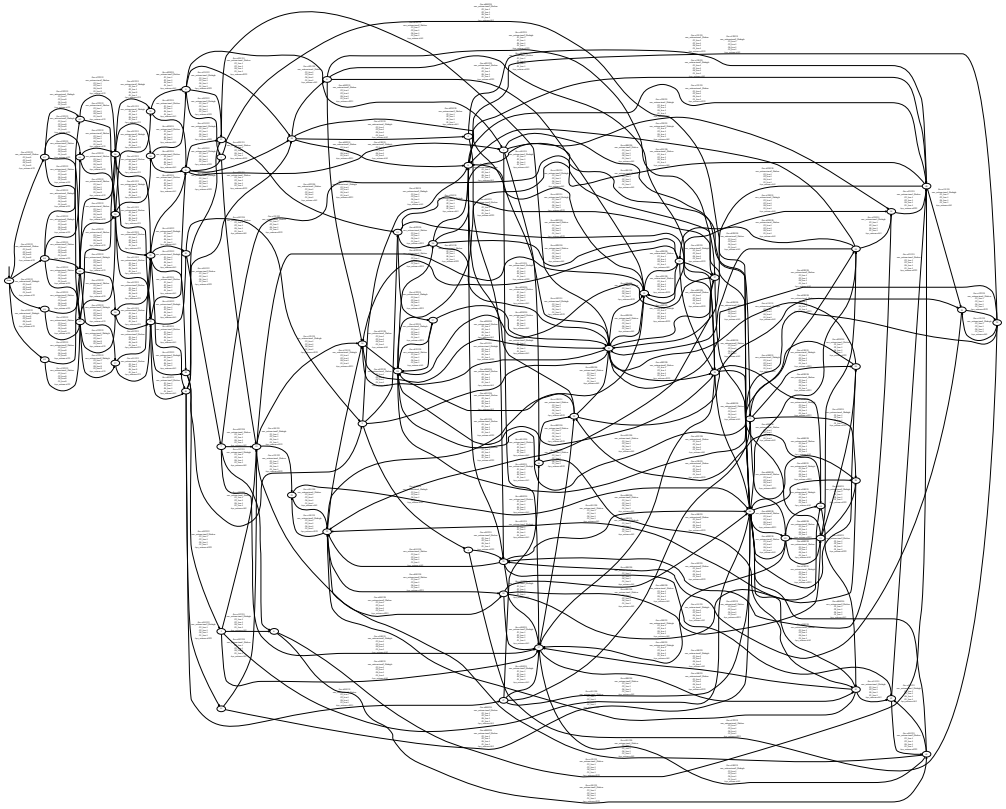

Figure 2.4: Synthesized controller by TuLiP for concrete traffic network

in the concrete traffic network, either. Actually TuLiP failed to synthesize a controller in the concrete traffic network for this modified GR(1). Therefore, over-approximation could give conclusive guidance on the concrete system control synthesis if the GR(1) specification is not realizable. Otherwise, over-approximation is not informative and we need look at the concrete system itself.

2.5 Integration of traffic discrete control into AIMS

In an AIMS with autonomous vehicles, traffic design specifications have to be met under the control of the discrete traffic controller. For example, the number of vehicles on main roads must eventually stay low. Chapter 2 presents an approach to synthesize a discrete controller for a traffic network in TuLiP. To meet the specifications, the synthesized

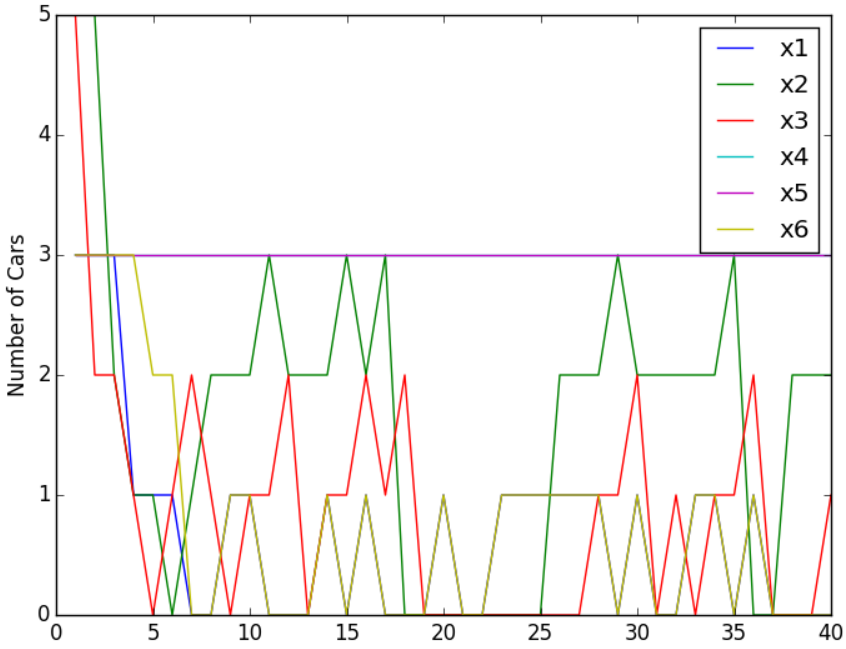

Figure 2.5: Concrete traffic state evolution

controller must automatically select the proper control in response to the real-time traffic. In this particular work, a proper control is to choose the right set of roads to be activated at each time step.

When designing an AIMS for a traffic network, desired traffic specifications are proposed based on emprical assessments. It is possible that those proposed specifications are too restrictive to be realized for the traffic network of interest. An over-approximated traffic network, which requires far less computation, is applied to suggest an infeasibility if the proposed specifications are unrealizable. If the specifications are approved in the over-approximation, same specifications are used to synthesized a discrete controller in the concrete traffic network.

The synthesized controller for the concrete traffic network can be integrated into the AIMS as a look-up table. With the traffic data available, the proper traffic control for the

current time step can be looked up in the synthesized controller. If a road is activated by the discrete controller, autonomous vehicles on the road pass through the intersection of interest under the continuous sliding-mode vehicle control, which will be presented in Chapter 4.

Chapter 3: Implementation of AIMS with Model-Based Assertions

The discrete controller derived in the last chapter, if implemented, on a real traffic network, would require much more effort than a research demonstration. To propose a typical structure of AIMS, discuss key features, and demonstrate practical designs of model-based assertions, a simple AIMS with two robot vehicles is studied in this research. This simple AIMS consists of an intersection, an intersection manager, vehicles approaching the intersection of interest, and a wireless network for communication. Ideally each local intersection manager distributes commands generated by the synthesized discrete controller which properly handles global traffic network dynamics. But for such a simple AIMS with only one intersection, there is not much traffic network dynamics involved. Therefore a "first come, first served" (FCFS) principle could be a good substitution for the discrete traffic controller, and can be applied to assign right of way in an implementation of AIMS. One major challenge for AIMS design is to reduce computational cost so that the AIMS can meet the real-time requirement [15]. Vasirani and Ossowski conclude that FCFS is quite efficient as a simple procedure by comparing different AIMS protocols, particularly for low traffic loads [16]. Furthermore, according to the simulation work done by Huang, Sadek, and Zhao, the FCFS reservation protocol offers "significant mobility and environmental benefits" compared to traditional intersection control methods [17]. Consequently, a FCFS reservation method

is adopted in this AIMS implementation.

The manager collects information from all approaching vehicles that are within a certain distance from the intersection of interest. This distance threshold that defines the responsibility region of a local manager primarily depends on the basic wireless network for communication. In contrast, the dedicated short-range communication technology (DSRC) in the 5.9 GHz band is specially for vehicular environments in the U.S. and has been recognized by many authors in the study of V2V (vehicle to vehicle) and V2I (vehicle to infrastructure) communication [18, 19, 20]. In this laboratory research, a basic wifi communication is sufficient. In addition, the threshold distance is also related to speed limit and traffic load. For example, a larger distance threshold should be selected for intersections connected to highways to allow enough processing time for both the manager and the approaching vehicles, while a smaller distance threshold is more reasonable for intersections with heavy traffic load to avoid computational saturation. In this study the threshold distance is not defined explicitly because there are only two robot vehicles involved within a very limited space. Robot vehicles are connected to the manager upon start. The manager updates registered vehicles periodically and discards vehicles that have already left the intersection of interest.

3.1 AIMS in this laboratory research

3.1.1 The intersection

This study uses a four-way intersection with a size that ensures enough space for the ground robot vehicle to pass through. Each direction has an entry lane and an exit lane. The width of a single lane should be a little larger than the vehicle width. Specifically the track width is 0.36m here, thus the intersection lane width is taken to be 0.5m. This can be justified by comparison to the design of bus lane widths in URBAN INTERSECTION DESIGN GUIDE, which requires no narrower than 3.7m width given a maximum bus width

of 3.2m [21].

The intersection in this research is defined by reflective tape adhered to the ground (Figure 3.1), which can be detected by an optical sensor to determine when a vehicle enters or departs from the intersection of interest. Outside of the enclosing tape are four two-way roads. There is no conventional traffic light or stop sign for this intersection.

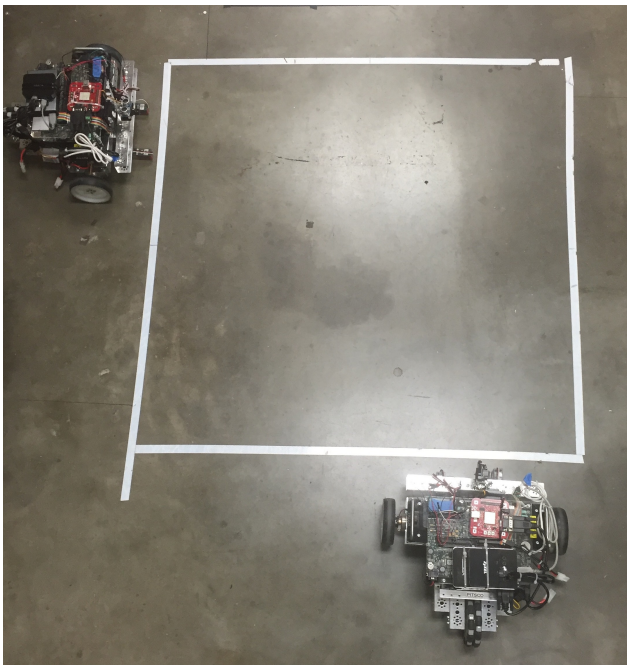

Figure 3.1: Intersection in this laboratory research

3.1.2 The ground robot vehicle

The robotic vehicle used in this research is the DaNI 2.0, a differentially steered ground robot vehicle manufactured as an education and research product by National Instruments, Inc. (Austin, TX). It has two independent motor-driven wheels on front axis and one omni-directional wheel in the rear. DaNI comes standard with a PING sensor that can span using a servo motor, and two encoders providing wheel motion information. DaNI is controlled by a single-board real-time processor (sbRIO-9632) made by National Instruments, which can be

programmed and interfaced using LabVIEW. The LabVIEW Robotics Module facilitates the construction of real-time and FPGA applications on DaNI. The single-board RIO integrates a 400 MHz Freescale real-time processor, a 2M gate Xilinx Spartan FPGA, 110 digital input-output (DIO) lines, 32 integrated analog input (AI) channels, 4 integrated analog output (AO) channels, an Ethernet port to connect with other devices, and a RS232 serial port for peripheral devices. The entire DaNI can be battery-powered (e.g., a 12-volt rechargeable NIMH battery from TETRIX) or by a AC-DC adaptor during benchtop development. Two Pitsco 12 VDC permanent magnet DC motors are independently controlled by a dual DC motor speed controller from Dimension Engineering. Motor speeds are adjusted by the motor controller which accepts pulse width modulation signals generated from DIO lines on the single-board RIO. Pictures of DaNI and its block diagram is present in Figure 3.2.

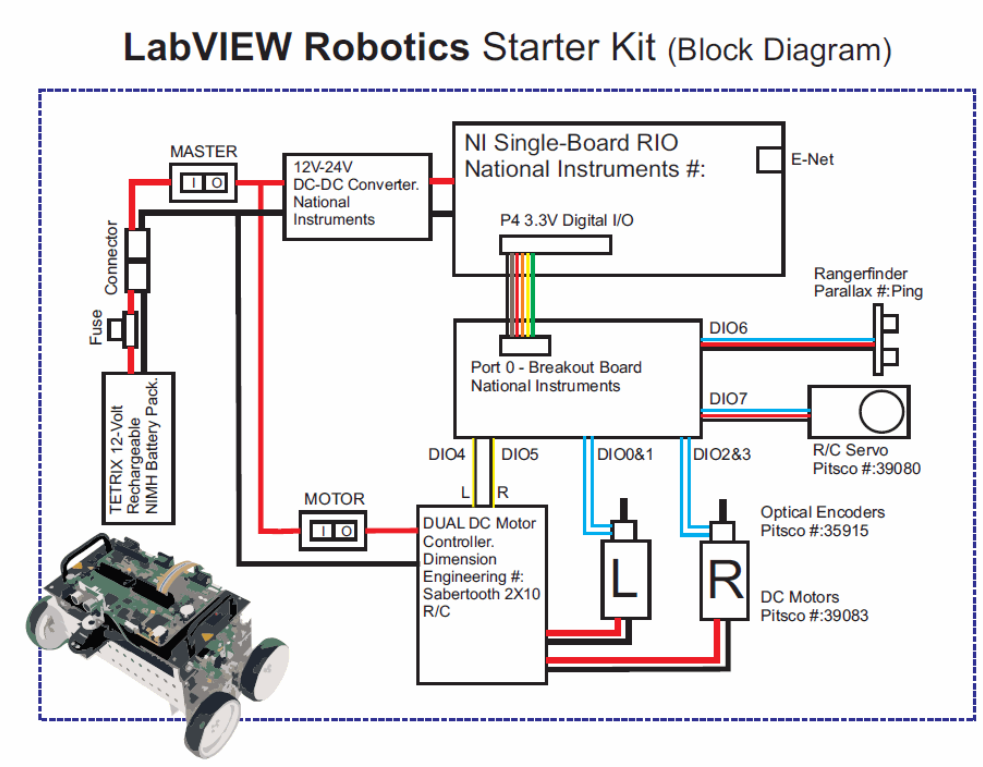

Figure 3.2: Operation block diagram of DaNI (ni.com)

3.1.3 The intersection network

A Wireless-G broadband Linksys router from Cisco is configured as an Access Point to create a local intersection network. The intersection manager, which is a laboratory desktop, connects to the same subnet via a wireless USB adaptor. Each DaNI has an Ethernet port which can connect DaNI to the same subnet. To do this, a ZyXEL MWR102 Travel router is switched to Client Bridge Mode and connects its LAN port to the Ethernet port on DaNI. As a result, a connection between DaNI and the remote access point created by the Linksys Router is established. DaNI is now able to communicate with the manager since they are within the same subnet. Each ZyXEL router is powered by its own power bank to prevent draining power from the DaNI. An IP is needed for a network node to communicate with other nodes. In this setup, static IP addresses are assigned to the manager and to the two DaNIs. In practical intelligent transportation systems, due to a rapid switch between different access networks in vehicular environment, IP mobility support needs to be carefully addressed [22]. Several IP mobility support solutions are discussed by Banda, Mzyece, and Noel [22].

3.1.4 Sensors on DaNI

Except for the integrated PING sensor and encoders, DaNI can interface with many other sensors through its DIO lines, AI lines, and serial port. In this research, an Inertial Measurement Unit (IMU) and two optical line sensors are used along with the encoders.

Inertial measurement unit. A VN-100 Inertial Measurement Unit (VectorNav Technologies, Dallas, TX) can provide attitude, angular rate, and acceleration of the DaNI if they are rigidly bundled together. This IMU uses a right-handed coordinate system with z-axis pointing downward and y-axis pointing to the right. It is powered by the same power bank

that powers the ZyXEL router. The IMU has two communication modes: UART Mode and SPI Mode. In this research, DaNI communicates with the IMU using UART Mode through a RS232 serial port. The baud rate is configured to be 115200. With a proper command, the IMU continuously captures and outputs attitude data to the single-board RIO at a frequency of 50 Hz. The TARE command for the IMU can reset the current yaw angle to 0 degree and displays subsequent yaw angles in reference to the current orientation.

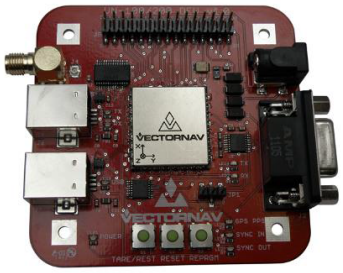

Figure 3.3: VN-100 inertial measurement unit (vectornav.com)

Wheel encoder. Two incremental rotary encoders (Figure 3.4) are installed on each motor axle to estimate wheel speeds. These encoders have 400 pulses per revolution and feature a ±1 rad/s resolution with a16-ms clock cycle. By counting the number of pulses over a timed period, encoder can be used to estimate wheel speed.

Figure 3.4: Incremental encoder kit on DaNI (pitsco.com)

Optical line sensor. The 05A05 line tracking sensor from VEX (VEX Robotics, Greenville, TX) distinguishes a highly reflective area from the surrounding surface and thus is able to detect the reflective tape enclosing the intersection of interest. The line sensor illuminates a surface with infrared light and receives the reflected infrared radiation. According to the intensity of the reflected radiation, this line sensor reads different voltages. In particular, the sensor outputs a low analog voltage when the surface is highly reflective and high analog voltage for absorptive surfaces. To avoid saturating the infrared sensor, line sensors are mounted onto the DaNI with a distance of 3 mm from the ground. Line sensors are powered by 5 VDC power output from the single-board RIO and the data is captured by analog input channels.

Figure 3.5: Optical line sensor (vexrobotics.com)

3.2 Program architecture

The entire AIMS based on a FCFS reservation protocol is programmed as a real-time project in LabVIEW. The programming mode is configured to be a FPGA interface so that customized high-speed FPGA control can be integrated into the project. The FPGA module in LabVIEW can handle data transfer between the real-time processor and FPGA and facilitate construction of custom FPGA programs. This LabVIEW project consists of a host computer and two real-time targets, all of which are network nodes within the dedicated

local wireless network. The host computer plays the role of intersection manager while the real-time target on DaNI controls the robot vehilce. The Host Main.VI running on desktop is in charge of the intersection manager. It collects data from the DaNIs and distributes commands to each DaNI. The Target DaNIx.VI (x represents the index of a DaNI) receives commands, acquires data,handles DaNI dynamics, and communicates with other DaNIs. Communication and collaboration among different network nodes are rather important for AIMS. In this AIMS, network-published shared variables and single-process shared variables with real time first in first out buffer (RT FIFO) enabled provide network and inter-task data transmissions respectively. Network communication refers to data transfer across network nodes and thus network-published shared variables have to be published across the AIMS network. Inter-task communication shares data among different loops within one network node and thus are deployed only locally on the DaNI. The program architecture of this AIMS with two robot vehicles and model-based assertions is depicted in Figure 3.6. More robot vehicles can be readily integrated into the system in a similar way. The entire program can be divided into four segments: initialization, a reservation-assignment loop on the intersection manager, a low-priority loop for monitoring and updating vehicle status on DaNI, and a high-priority loop for vehicle motion control on DaNI. The processor of the embedded controller on each DaNI dedicates its resources to maintain a time-critical feature for the high-priority loop while the low-priority loop only takes effect when the high-priority loop is in idle status. Each segment of the program is detailed below.

The status and motion plan of DaNI are recognized as the core data that need to be shared. Each approaching DaNI carries an information package that contains the arrival time, approaching speed, IP address, command, proposed motion plan, arrival status, and passed status. A cluster in LabVIEW bundles different data types into one single data type and is used for the vehicle information package. Arrival time is the time taken for a DaNI to arrive at the intersection of interest after being registered with the manager. The intersection

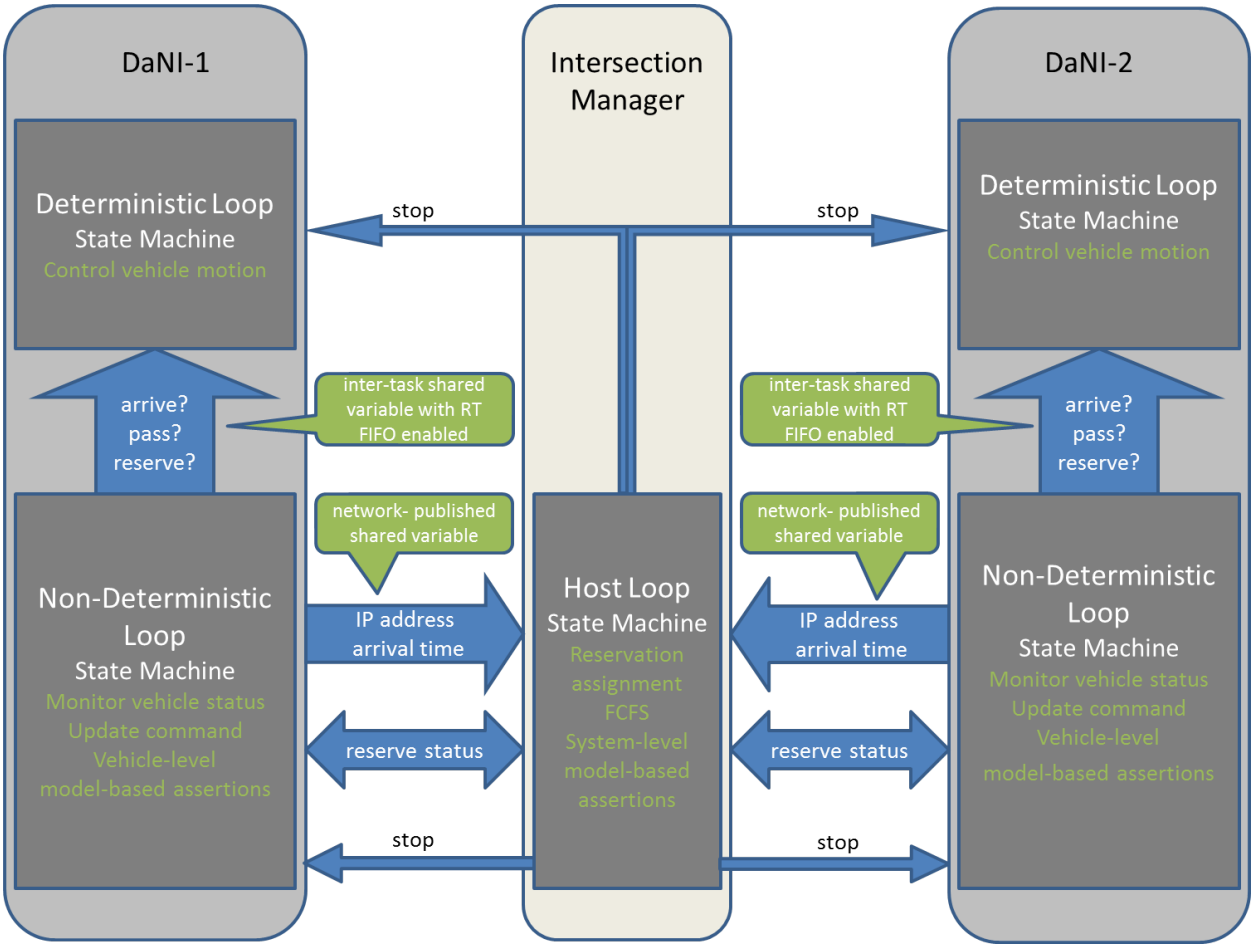

Figure 3.6: Program architecture of the AIMS using a FCFS reservation protocol

manager reorders all approaching DaNIs based on arrival time and assigns a right of way on a FIFS basis. Arrival time has to be the first element of the vehicle information package so that the package can be compared directly. This is because LabVIEW orders clusters according to their first elements and only compares subsequent elements if all preceding elements are equal. To assign a right of way, the manager identifies a specific DaNI via IP address and then updates the reservation status associated to that IP address. Each DaNI either stops or proceeds based on its reservation status. A STOP command stops a DaNI at the intersection of interest upon arrival. A CONTINUE command promotes a DaNI to get into the intersection. Arrival status notifies the manager whether a DaNI arrives at the intersection and is ready to be assigned a right of way. In the case where a DaNI reserves a right of way upon arrival at the intersection, it proceeds without a stop. Once in the intersection, a DaNI may go straight, turn left, or turn right according to its motion plan. Different motions are realized by the motion control over the motor-driven wheels. Once a DaNI exits the intersection, its passed status notifies the manager to release the right of way to other DaNIs. Vehicle information package can be accessed globally as a network-published shared variable named "Car Information-net", and internally as a single-process shared variable named "Car Information-RT". The last shared variables for this AIMS is a network-published stop command, which is used to stop the whole AIMS manually.

3.2.1 LabVIEW programming environment

LabVIEW is a graphical development environment from National Instruments. As a visual programming language, LabVIEW is easier to be interpreted by engineers from fields where script programming is not used. LabVIEW replaces conventional scripts with blocks, which are called visual instruments. Each LabVIEW program has its own front panel for user interfaces and block diagram for program construction. User inputs and program outputs are, respectively, handled by controls and indicators. LabVIEW is able to interface with

various hardware from National Instruments, such as an embedded industrial controller or a data acquisition device.

3.2.2 Initialization

At the start of an AIMS run, both the intersection manager and DaNIs have to initialize all controls to their default values before proceeding to the subsequent segments of programs. During an initialization, the intersection manager also collects vehicle information package from each approaching DaNI and synchronizes DaNIs through a triggering mechanism. The intersection manager reorders DaNIs based on arrival time.

For DaNI, first of all, it initializes shared variables. Each DaNI creates a reference to the serial port connected to the IMU, configures the serial port, and sends a request for continuous data acquisition. It also creates a robotic FPGA session to use built-in robotics VIs that interface with the FPGA. These LabVIEW specific initializations would take different forms for other types of platforms.

3.2.3 Loop on the intersection manager

To monitor the reservation status of the intersection of interest and pass on a right of way to the next DaNI, a loop containing a finite state machine is applied to handle the transitions of right of way. The finite state machine includes an assignment state for each approaching DaNI and a monitoring state managing the transition of right of way from one DaNI to another. The intersection manager reorders DaNIs into a reservation assignment sequence. The first DaNI is assigned a right of way and then the finite state machine transitions into the monitoring state. At the same time, the index of DaNI to be assigned a right of way updates to 2. In the monitoring state, intersection reservation status of DaNI #1 is continuously monitored. The authorized DaNI, which is DaNI #1 here, releases its right

of way after it passes through the intersection. Next, the current state transitions into the assignment state for DaNI #2 so that a right of way is reserved for DaNI #2. The subsequent DaNIs receives a right of way in the same way. Figure 3.7 demonstrates a transition of right of way from DaNI-i to DaNI-j. At one point, if all DaNIs in the reservation assignment sequence have passed through the intersection of interest, this finite state machine will be triggered to exit and the intersection manager is ready for a new collection of vehicles.

Figure 3.7: Finite state machine for intersection reservation assignment

3.2.4 Low-priority loop on DaNI

A low-priority loop with a finite state machine on each DaNI keeps a track of sensor measurements in order to update DaNI status and assertion status. Approach, pass, and exit are taken as the three states of the finite state machine within this low-priority loop. A DaNI stays at the approach state until arriving at the intersection after being registered with the intersection manager. Transition from approach to pass state is triggered by the detection of reflective tape that defines the intersection of interest. Note that a DaNI only enters the intersection if the intersection is not occupied. Two line sensors are mounted in the front of DaNI in parallel. One line sensor is ahead of the other by about 1 centimeter. This configuration is to distinguish between an entrance and a departure. As a DaNI enters the intersection, the line sensor placed ahead will detect a reflective surface. And shortly after that the front sensor will detect a normal surface while the other detects a reflective surface. At this particular point, two line sensors jointly determine when a DaNI arrives at the intersection. In the other case where a DaNI departs from the intersection, the line sensor placed ahead will detect a reflective surface first and indicate a departure. Positions

of the two line sensors upon arrival and exit are depicted in Figure 3.8. The left layout represents the case for an entrance and the right layout represents a departure.

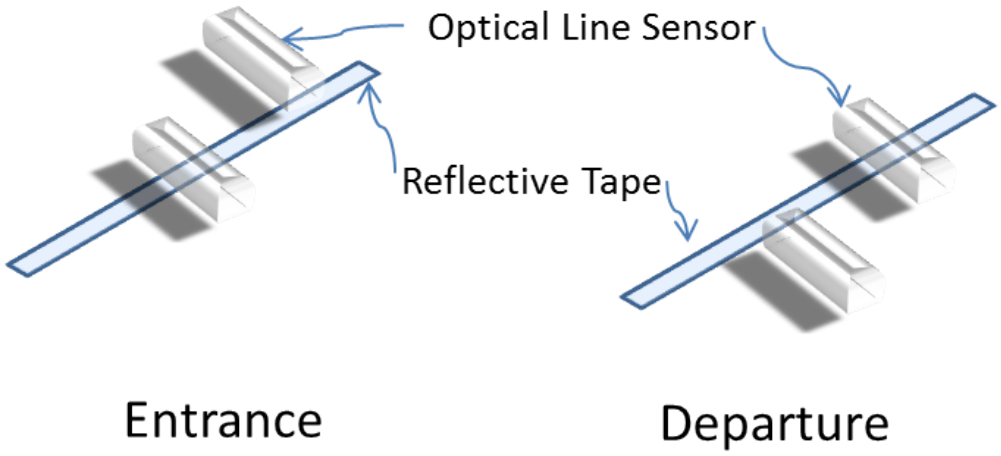

Figure 3.8: Line sensor layout upon entrance to and departure from the intersection

After a DaNI is authorized to pass the intersection, its reservation status is monitored. Once assigned a right of way, the DaNI is promoted to get into the intersection under motion control. Simultaneously the arrival status of the DaNI becomes TRUE. As the DaNI passes through the intersection, the exit condition shown as the right layout in Figure 3.8 is examined. If exit condition is met, the current state transitions into exit, where the DaNI releases its right of way and updates its pass status to TRUE.

3.2.5 High-priority timed loop on DaNI

A high-priority timed loop with a finite state machine is employed on each DaNI to control its motion. A high-priority timed loop in LabVIEW features a fixed loop period, which is set to 20 milliseconds in this application. In LabVIEW, a high-priority timed loop is ensured a time-critical feature by setting its loop priority to the highest value within the target. In this way the onboard processor will dedicate its resource to executing the

high-priority loop in order to maintain a fixed loop period. The finite state machine in this motion control loop triggers state transitions by reading motion commands from the low-priority loop. With a stop command, DaNI stops at the intersection of interest upon arrival. With a continue command, DaNI passes through the intersection according to its motion plan. Once the DaNI exits the intersection, it continues going straight for another 1.8 seconds so that the next DaNI will not run into it. This is only necessary for laboratory runs. In real cases, a vehicle will move forward on its planned road after exiting an intersection.

The motion control is designed based on a kinematic model of the DaNI without slipping. Since the center of gravity of DaNI roughly lies on the center of front track, we can assume DaNI does not exhibit any transverse motion. In a body-fixed reference frame as depicted in Figure 3.9, the kinematic equations describing the motion of DaNI can be denoted as,

$$\begin{aligned} v_x &= \frac{R_w}{2}(\omega_{left} + \omega_{right}) \\ v_y &= 0 \\ \omega_z &= \frac{R_w}{B}(\omega_{left} - \omega_{right}). \end{aligned} \tag{3.10}$$

where R_w and B are the wheel radius and track width of DaNI. ω_{left} and ω_{right} are rotational speeds of two front wheels.

According to the kinematics above, a motion control maintains equal speed for both wheels if a DaNI wants to go straight. A left or right turn is realized by differential wheel speeds such that DaNI rotates about a center. In order to do a rotation about a point, DaNI has to move forward for a certain distance before making a turn, otherwise the start turning point will be not symmetric to the end turning point about the corner of the intersection. This distance was measured to be 7.6 cm. Three possible trajectories of a DaNI under this simple control approach are depicted in Figure 3.10.

To estimate the wheel speed difference for a left turn, assume the speed difference is $2\Delta\omega$ and the turning radius is R_{left} . Note that the total time used during a single motion,

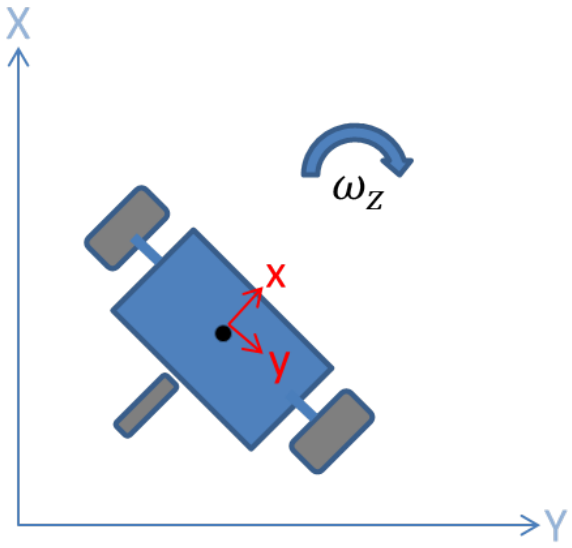

Figure 3.9: Body-fixed reference frame for DaNI

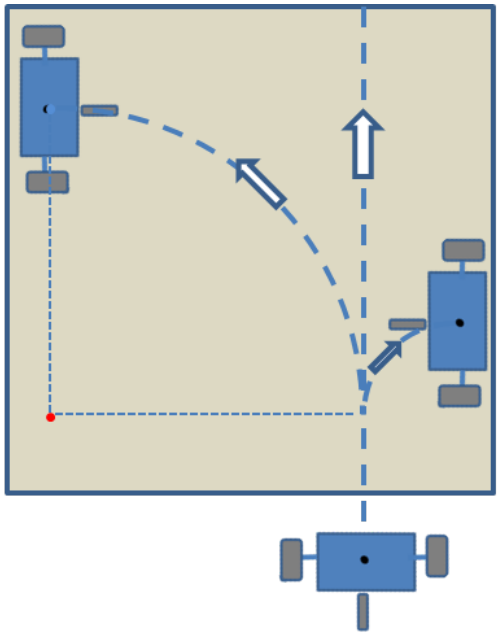

Figure 3.10: Possible trajectories of a DaNI within an intersection

no matter with respect to traveling distance or rotational angle, stays the same. Thus, two equations are generated as,

$$\begin{aligned} \frac{1}{2}R_w((\omega_{left} - \Delta\omega) + (\omega_{right} + \Delta\omega)) \cdot t &= R_{left} \cdot \frac{\pi}{2} \\ \frac{R_w}{B}((\omega_{left} - \Delta\omega) - (\omega_{right} + \Delta\omega)) &= -\frac{\pi}{2}. \end{aligned} \tag{3.11}$$

Given that left and right wheel speeds are the same before turning ($\omega_{left} = \omega_{right} = \omega$),

$$\Delta\omega = \frac{\omega B}{2R_{left}}. \tag{3.12}$$

With the designed turning radius, $\Delta\omega$ is 0.29ω . In real test, an effective value was found to be 0.30ω. For a right turn, it turned out that a desired right-turn path could be achieved by simply setting the right wheel speed to 0.

A major problem encountered while designing motion controller is that the resolution of a wheel speed command is too large. According to a wheel speed control test, the revolution is 1 rad/s. A speed command within ±0.5 rad/s around a nominal value such as 5 rad/s is automatically rounded to the nominal value. This can be explained jointly by the 1 rad/s encoder resolution and that the feedback PID wheel speed control algorithm is fundamentally based on the encoder estimation of the wheel speed. As a result, the motion control implemented here, while adequate for demonstrating AIMS as a cyber-physical system, is far from accurate. Therefore, a robust sliding-mode controller is developed in the next chapter of this thesis.

3.3 Assertions and tests

Both logical and physical models are employed for constructing assertions that are integrated into the AIMS. Though the underlying models applied to construct assertions can vary, those assertions share the same form, which is comparing the actual status or performance measures to expectations derived from models. If they match, we say the

assertion passes; otherwise, the assertion fails. If any assertion fails, corresponding assertion failure information is sent to the user and every system component. Then, the whole system exhibits a programmatic stop or other desired action. The notification message reveals the type and location of the assertion failure. If an assertion failure stops the whole AIM system, only the earliest failure will be displayed. Unlike a user-input forced stop, a programmatic stop will not destroy the real-time and concurrency nature of a cyber-physical system. For example, messages can accumulate at a serial port during the interrupt of a user input and lead to an error. Details for each assertion and tests designed to validate them are discussed below.

3.3.1 Initialization check

Network-published shared variables are primarily used in the AIMS to share information among different components. At the start of the AIMS, those variables are supposed to be initialized to default values. This initialization process does take some time, so it is possible for a shared variable to be read before being initialized. In that case the program will execute with incorrect shared variable values and can result in serious consequences. For instance, in the case where while loops are used to synchronize the start of all robot vehicles, the Boolean shared variable named "start" controlling the while loop on a robot vehicle may start the vehicle earlier if "start" is not initialized back to false. Therefore, an assertion checking the initialization process is necessary for the AIMS.

This assertion check keeps comparing the read values to initialization values within a while loop until they all match or a timeout occurs. The read value for a shared variable is the current value stored in the memory. The initialization process writes a default value to each shared variable. A timeout event indicates certain variables are not able to be updated within a reasonable time. The initialization assertion fails only when timeout occurs. Two tests were designed. With timeout value set to 1000 ms, the AIM system proceeded with

successful initialization of shared variables. If set to 10 ms, one of the robot vehicles failed to initialize all its shared variables within 10 ms and resulted in an initialization timeout assertion failure. This assertion failure was displayed as "Initialize Check" on the front panel and promoted the whole system to a stop.

3.3.2 Ambient light check

The intersection of interest is defined by reflective tape on the ground. Detection of entrance to or departure from the intersection by each robot is based on optical sensor readings. An optical sensor detects the tape by sensing a higher reflected light intensity. As a result, keeping the same ambient light condition is critical to the performance of the whole AIMS. Failing to respond to distinct variations in ambient light can cause malfunction of the AIMS.

An optical sensor reads a small nominal range under regular light condition without reflective tape. This assertion takes the current sensor reading upon start of the AIMS and compares it to the nominal value. If this reading falls outside the nominal range, the ambient light check assertion fails before the robot vehicles start moving. Two tests were conducted to validate this assertion. In the first test, the AIM system ran as expected without any assertion failure. In the second test, a reflective tape was placed at the start position of a DaNI. The test sensor reading was below the nominal range so that the assertion for ambient light check failed.

3.3.3 DaNI stop check

Any robot vehicle entering the intersection of interest that does not reserve a right of way is supposed to stop in front of the intersection to wait for the authorized vehicle. If any unauthorized robot vehicle does not stop properly, collisions may occur. Thus, an assertion

is necessary to check the velocity of those robot vehicles to assert whether they are stopped. Wheel speed measured by two integrated encoders on the DaNI can provide an estimate of vehicle velocity if there is little or no ground slip.

Note that the robot vehicle does not stop immediately upon being commanded to stop. To prevent inaccurate assertion check results, wheel speed is checked only after some delay once the DaNI is commanded to stop. In the current system, the encoders have an uncertainty of 1 rad/s, so the wheel speed is compared to 1 instead of 0 rad/s. If the wheel speed still exceeds 1 rad/s after the time delay, the assertion will fail. Two experiments were conducted to test this assertion. The time delay was first set to 400 ms. The AIM system ran correctly. As the time delay was decreased to 200 ms, the robot vehicle reserving no right of way was unable to stop within 200 ms after 200 ms. Therefore, the program asserted that the unauthorized robot vehicle failed to stop at the intersection.

3.3.4 DaNI not responding

The wifi connection status is monitored by LabVIEW built-in features and will display an alarm if lost. However, even with wifi connection, the real-time program running on each robot vehicle may stop itself for various reasons. In this scenario, the robot vehicle may stop at the center of the intersection and reserve the right of way forever, so the AIMS collapses. An assertion monitoring the running status of programming on each robot vehicle can address this issue.

This assertion is realized by continuously monitoring the update of a particular network-published shared variable deployed to each robot vehicle. Within the time-critical loop for each robot vehicle, a Boolean variable named "running status" continuously switches its value between false and true. At the manager side, if any of those "running status" variables is not changing its value within 1000 ms, the manager will assert that the robot vehicle is not running anymore and stop the whole AIMS. In order to simulate stopping the program

on robot vehicles without losing wifi connection, the program can be stopped manually. Two tests were run to examine this assertion. First, without stopping any robot vehicle program manually, the AIMS ran without an assertion failure. After stopping the program on one of the robot vehicles, the "running status" variable for that vehicle stopped updating. After 1000 ms, the manager asserted a DaNI was not responding.

3.3.5 Undesired motions check

Each robot vehicle can pass through the intersection of interest in three ways: go straight, turn left, or turn right. Motion of robot vehicles within the intersection is purely determined by the motion controller. To evaluate the motion performance, an assertion can be integrated into the program to assert whether a robot vehicle has moved according to the motion plan. Criteria in terms of passing time and turning angle should be applied to evaluate the motion of a robot vehicle. Based on a vehicle kinematic model, theoretical time required to pass through the intersection can be estimated given the wheel speeds. If the actual passing time is apparently shorter, the program should assert a motion control failure to prevent any safety issue. For example, a robot vehicle may stop if there is any reflected tape within the intersection. If not properly handled, this can lead to a collision.

Yaw angle for each robot vehicle is measured by the integrated IMU. Turning left or right requires that the change in yaw angle before and after turning falls into a tolerance region around $\pm 90^o$. Going straight requires that the deviation in yaw angle is very small. Owing to the inherent dissimilarity between the two motor controllers on the DaNI, the vehicle does not go straight even with the same speed command. If the variation in yaw angle falls outside of the expected range, the assertion should indicate a failure and stop the whole AIMS.

Time when a robot vehicle starts passing through the intersection is saved to a local variable. The pass duration is calculated once the robot vehicle exits the intersection. At

the same time, variation in yaw angle is also computed. If any value is far away from its theoretical range, this assertion will indicate an undesired vehicle motion on the AIMS front panel. The theoretical variations in yaw angle are respectively $0^o \pm 7.5^o$, $-90^o \pm 15^o$ and $90^o \pm 5^o$ for a straight motion, a left turn, and a right turn. Eight tests were conducted to validate this assertion. Test results are tabulated in Table 3.1. It is shown that the assertion failed when the tolerance of yaw angle variation was set to be smaller or a reflective tape was placed within the intersection.

Table 3.1: Test results for validation of vehicle motion assertion

Motion	Test Scenario	Assertion Result
Straight	Reflected tape placed within the intersection	Fail
	No tape within the intersection	Pass
	Yaw angle tolerance set to $\pm 5^o$	Fail (-6.772^o)
	Yaw angle tolerance set to $\pm 10^o$	Pass (-5.649^o)
Left	Yaw angle tolerance set to $\pm 10^o$	Fail (-76.744^o)
	Yaw angle tolerance set to $\pm 15^o$	Pass (-78.199^o)
Right	Yaw angle tolerance set to $\pm 2^o$	Fail (86.589^o)
	Yaw angle tolerance set to $\pm 5^o$	Pass (87.119^o)

3.4 AIMS: laboratory study to practical implementation

The simple four-way intersection and DaNI robot vehicles successfully demonstrated the structure of a laboratory AIMS and associated model-based assertions. An intersection manager, approaching vehicles, and an information exchange system form a basic AIMS. A reliable real-time data exchange mechanism, as required by collaborations among vehicles and between vehicles and the intersection manager, is essential to the AIMS. In a practical AIMS, while the structure remains similar, an advanced discrete controller, instead of the FCFS, would be required to control and maintain the specifications of a traffic network with multiple intersections. Instead of a local wireless communication, a dedicated short-range communication technology (DSRC) should be used for V2V and V2I communication.

System-level and component-level model-based assertions monitor and validate AIMS performance and trigger a programmatic stop if any assertion fails. Peripheral sensors on DaNI capture useful data that can be interpreted to evaluate model-based assertions. The resolution and speed of those sensors are critical to generating informative and accurate assertions. In practical implementation, reliable, accurate, and cheap industrial sensors have to be applied to continuously collect data for model-based assertions as well as any closed-loop control. In addition, proper actions have to be taken in case of an assertion failure since a system-wide programmatic stop is obviously not desirable for a real autonomous traffic network.

Chapter 4: Robust Trajectory Control for DaNI

An autonomous intersection management system with autonomous vehicles can be modeled as a hybrid control system. The intersection manager acts as a discrete planner that assigns discrete motion plans to each vehicle. An autonomous vehicle implements the discrete motion plan while exhibiting continuous dynamics. For example, before entering the intersection, a vehicle may navigate at a certain speed. On entering the intersection, the intersection manager may update the discrete plan to conduct a right turn between t_i and t_{i+1} according to its discrete control strategy and the vehicle's request. Then the vehicle will take this updated plan and implement a right turn at the authorized time.

A continuous trajectory controller is required to handle low-level vehicle dynamics and environmental constraints while implementing discrete plans. As discussed in previous chapters, even with a simple discrete control strategy, an open loop trajectory control for DaNI may fail to fall in an acceptable accuracy. Therefore, in the application of autonomous intersection management with autonomous vehicles, a robust vehicle trajectory control is essential to implementation of high-level commands. In this chapter, a Sliding-Mode Controller (SMC) is proposed for trajectory control.

A typical SMC controller for a nonlinear system (NLS), such as the DaNI dynamics, is shown in Figure 4.1. A sliding-mode controller is a type of robust controller that can

deal with complicated nonlinear systems. In Figure 4.1, ELO is an extended Luenberger observer, PF represents a pre-filter that is used to estimate higher-order dynamics of the reference, u is the control inputs generated by the SMC controller, y denotes the outputs of the whole closed-loop system, r is the desired(reference) outputs, and ξ is the norm form. These elements will be discussed in detail below.

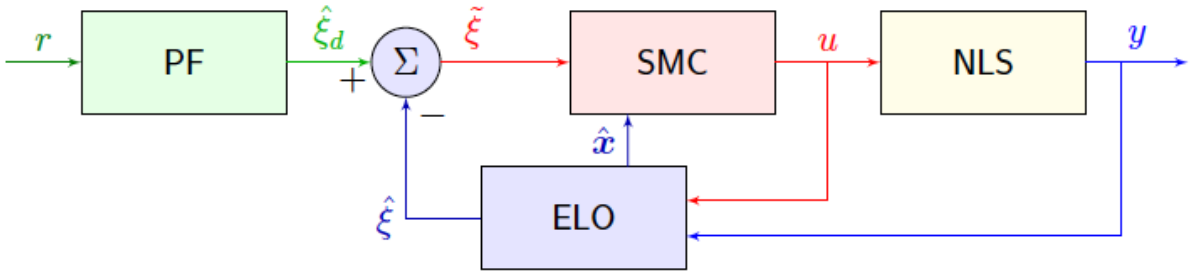

Figure 4.1: Schematic for a typical SMC feedback control system ([23])

As discussed in Chapter 2, at each time step the discrete traffic controller decides which vehicles are allowed to pass through intersections. Each authorized vehicle has to exit the intersection within the discrete time step determined by the discrete controller. For example, in a traffic network controlled by a discrete controller that features a 5-second time step, a currently authorized vehicle must complete the left turn within 5 seconds. Besides this overall time constraint, there are a couple of other specifications that the vehicle trajectory SMC controller has to meet. First of all, the controlled trajectory has to follow the desired trajectory with a defined accuracy. This can be achieved by restricting the steady-state error. Second, DaNI has to follow the desired trajectory without too much delay, which can be handled by specifying settling time in the control environment. Why is a low delay so important? In the Autonomous Intersection Management System studied in Chapter 2, high-level specifications that have to be satisfied to ensure safety and liveness of the traffic network are based on linear temporal logic. Therefore, maintaining a low delay in the continuous domain is essential to maintaining those temporal logic specifications in the

high-level discrete domain. For instance, if one vehicle receives a discrete plan to turn left within $[t_i, t_{i+1} \pm \delta t]$ while another vehicle approaching in an opposite direction receives a discrete plan to go straight within $[t_{i+1} + 5\delta t, t_{i+2} \pm \delta t]$, collision may occur if the first vehicle exhibits a delay larger than $5\delta t$. Third, states in the control model can either be readily measured or observed from measurements. This is verified in the following section. Fourth, typically only desired trajectory are given. Higher-order dynamics of the desired trajectory, if needed, must be accessible by the control model in order to build a controller. This is solved by adding a pre-filter. Lastly, the controller has to be robust. A vehicle operating in a practical environment such as a real intersection is expected to encounter disturbances and uncertainties. Parameters estimated for the 2-D model used to build the SMC controller are not guaranteed to be perfect. There are also bounded uncertainties associated with measurements. As a result, the controller designed must be able to deliver good trajectory control under bounded uncertainties and disturbances within design specifications.

4.1 Model

A 2-D model without wheel slip and in the body-fixed frame for the DaNI vehicle is applied here. This model of the DaNI was originally presented by Idler [11], but some changes have been made to incorporate the closed-loop SMC controller. The corresponding bond graph is shown in Figure 4.2. Motor inductance and battery resistance are assumed negligible. DaNI's motion is directly controlled by the net forces produced by both wheels. Each net force is determined by the corresponding wheel, drivetrain, and motor. Taking the right wheel as an example, the right motor takes pulse-width modulation mc_r determined by the SMC controller. The induced torque is applied as a traction force on the right wheel through the right drivetrain since there is not slip assumed. By deducting rolling resistant force from the traction force, we get the net force provided by the right wheel onto the vehicle.

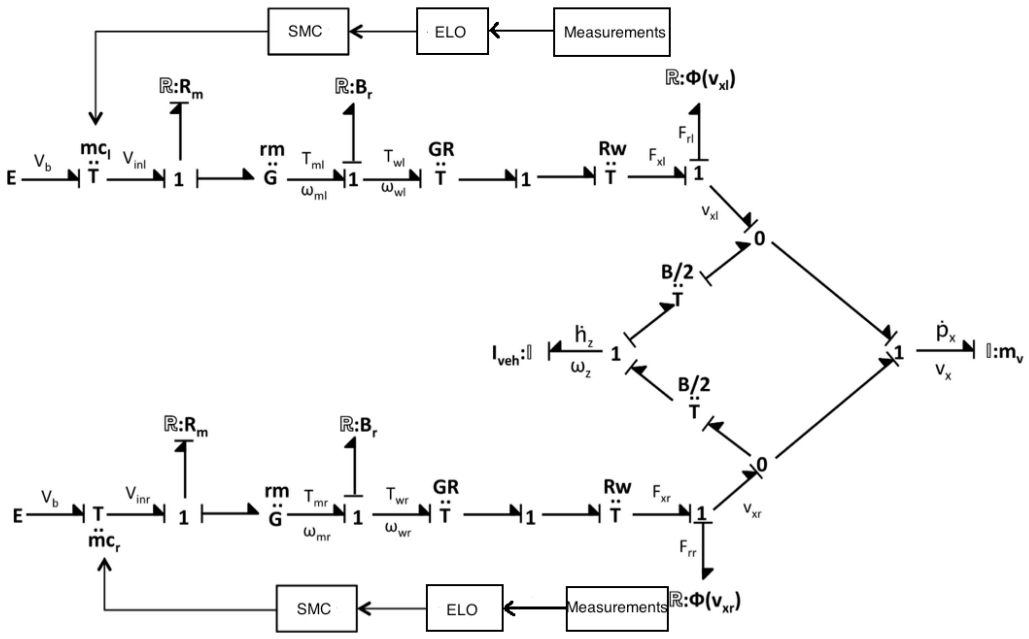

Figure 4.2: 2-D bond graph without slip for DaNI. SMC stands for sliding-mode controller; ELO stands for extended Luenberger observer; Measurements includes path length and yaw angle.

In this 2-D model, state variables are : $x = [r, v_r, \theta, \omega]^T$, where the DaNI longitudinal speed and accumulated path length are denoted by v_r and r. The vehicle yaw angle rate ω is the time derivative of the yaw angle θ. Since the accumulated path length is monotonically increasing, the mapping from a global position of DaNI (X, Y) to a body fixed coordinates (r, θ) is deterministic. Therefore we use $x = [r, v_r, \theta, \omega]^T$ as the control states in the SMC controller. Modulations for the right and left motor, $u = [m_{cr}, m_{cl}]^T$, are taken as inputs. The output is chosen to be $h(x) = [r, \theta]^T$. Path length can be measured by the encoders on wheels and θ can be measured by the IMU. According to the bond graph in Figure 4.2,

DaNI 2-D dynamics can be modeled by,

$$\dot{x} = F(x,u) = \begin{pmatrix} v_r \\ \frac{F_{xr}-F_{rr}+F_{xl}-F_{rl}}{m_{eff}} \\ \omega \\ \frac{F_{xr}-F_{rr}-(F_{xl}-F_{rl})}{I_{veh}}\frac{B}{2} \end{pmatrix}. \tag{4.13}$$

F_{xr} and F_{xl} are, respectively, the traction force provided by the right wheel and left wheel, and F_{rr} and F_{rl} are the rolling resistance forces. The effective mass m_{eff} lumps the DaNI mass as well as any significant internal rotational inertia of motor and drivetrain. The effective moment of intertia with respect to the yaw rotation is I_{veh}. The DaNI body is assumed to be an equilateral triangle whose center of mass is approximately located at the geometrical center. B is the track width. The values of m_{eff} and I_{veh} can be calculated through,

$$\begin{aligned} m_{eff} &= m_{DaNI} + \frac{2}{R_w^2}(J_{motor}GR^2 + J_{wheel}) \\ I_{veh} &= \frac{1}{12}m_{DaNI}B^2; \end{aligned} \tag{4.14}$$

where m_{DaNI} is the mass of DaNI, R_w is the wheel radius, GR is the gear ratio from motor to wheel, and J_{motor} and J_{wheel} are estimated moment of intertia for the motor and the wheel.

Based on the bond graph, expressions for the forces can be derived as in Eq.4.15. Note that forces are proportional to control inputs mc_r, mc_l, which implies that the system

dynamics can be written in affine form $\dot{x} = f(x) + g(x)u$.

$$
\begin{aligned}
F_{xr} &= \frac{GR}{R_w}\left(\frac{V_b r_m}{R_m} m_{cr} - \left(\frac{r_m^2}{R_m} + B_r\right)\left(v_r + \omega\frac{B}{2}\right)\frac{GR}{R_w}\right) \\
F_{xl} &= \frac{GR}{R_w}\left(\frac{V_b r_m}{R_m} m_{cl} - \left(\frac{r_m^2}{R_m} + B_r\right)\left(v_r - \omega\frac{B}{2}\right)\frac{GR}{R_w}\right) \\
F_{rr} &= f_r \frac{mg}{2} sign\left(v_r + \omega\frac{B}{2}\right) \\
F_{rl} &= f_r \frac{mg}{2} sign\left(v_r - \omega\frac{B}{2}\right)
\end{aligned}
\tag{4.15}
$$

V_b is the supply voltage to the motors, which is assumed to be a constant. This assumption, along with the assumption of no battery resistance, will result in an unlimited capacity of power supply, which is impractical and will be discussed in detail in the simulation section. In this model, r_m is the motor constant, B_r is motor mechanical resistance coefficient, R_m is the motor electrical resistance, and f_r is the rolling resistance coefficient. Numerical values for those parameters are summarized in Table 4.1. To design the SMC controller, we need

Table 4.1: Model parameters for the 2-D dynamic model of DaNI

GR	20
R_w	0.0508 [m]
B_r	6.4e-5 [$\mathrm{N \cdot m \cdot s/rad}$]
R_m	2.6374 [Ω]
r_m	0.036 [$\mathrm{N \cdot m/A}$]
V_b	12 [V]
B	0.3683 [m]
m	3.6 [kg]
g	9.81[$\mathrm{m/s^2}$]
f_r	0.02 [N/kg]
J_{motor}	6.7e-6 [$\mathrm{kg \cdot m^2}$]
J_{wheel}	6.45e-5 [$\mathrm{kg \cdot m^2}$]
I_s	4.55 [A]
R_b	0.106 [Ω]

to transform the state equations into affine form. According to Eq.4.13, the DaNI dynamics

can be modeled as,

$$\dot{x} = F(x,u) = f(x) + g(x)\cdot u,\ g(x) = \begin{pmatrix} 0 & 0 \\ \frac{GRV_b r_m}{m_{eff} R_w R_m} & \frac{GRV_b r_m}{m_{eff} R_w R_m} \\ 0 & 0 \\ \frac{GRV_b r_m B}{2I_{veh} R_w R_m} & -\frac{GRV_b r_m B}{2I_{veh} R_w R_m} \end{pmatrix},\ f(x) = F(x,u) - g(x)\cdot u. \tag{4.16}$$

4.2 SMC trajectory controller

This section provides the details for the SMC controller design. First, controllability and observability are vertified. Second, expressions for a SMC controller are derived. Third, a dynamic system for a pre-filter is established. Last, error dynamics with an extended Luenberger observer are analyzed.

4.2.1 Controllability and observability

To verify whether it is feasible to control the system with input u, the controllability condition for this SMC control system has to be studied. To verify that the system states can actually be extracted from the measurments $h_m(x)$, the observability condition has to be analyzed. Since the *sign* function is involved in the state equations, a nonlinear approach to study controllability and observability is required. Analysis of nonlinear systems highly relies on Lie algebra. In particular, Lie bracket and lie derivative are introduced as,

$$\begin{aligned} [f,g] = [ad_f^1, g] = \frac{\partial g}{\partial x}\cdot f - \frac{\partial f}{\partial x}\cdot g,\ [ad_f^k, g] = [f, [ad_f^{k-1}, g]] \\ L_f^0(h) = h,\ L_f^1(h) = \frac{\partial h}{\partial x}\cdot f,\ L_f^k(h) = L_f(L_f^{k-1}(h)). \end{aligned} \tag{4.17}$$

Controllability is determined by evaluating the rank of the controllability matrix,

$$Ctr = [g_1, \cdots, g_m, [g_i, g_j], \cdots, [ad^k_{g_i}, g_j], \cdots, [f, g_i], \cdots, [ad^k_f, g_i], \cdots]. \tag{4.18}$$

For this 2-D DaNI model, $g(x) = g$ is independent of the state x, which implies $[ad^k_{g_i}, g_j] = \mathbf{0}$. Therefore, a controllability matrix can be selected as,

$$Ctr = [g_1, [f, g_1], g_2, [f, g_2]]. \tag{4.19}$$

Ctr is rather complex so that it is only evaluated numerically in MATLAB and not explicitly shown here. Its rank turns out to be 4 and equals the order of the system. Thus this DaNI 2-D system is controllable.

The local observability condition can be verified by evaluating the rank of the observability matrix,

$$Obs = \begin{pmatrix} \frac{\partial L^0_f(hm_1)}{\partial x} \\ \cdots \\ \frac{\partial L^0_f(hm_p)}{\partial x} \\ \cdots \\ \frac{\partial L^{n-1}_f(hm_1)}{\partial x} \\ \cdots \\ \frac{\partial L^{n-1}_f(hm_p)}{\partial x} \end{pmatrix} \rightarrow Obs = \begin{pmatrix} \frac{\partial L^0_f(r)}{\partial x} \\ \frac{\partial L^1_f(r)}{\partial x} \\ \frac{\partial L^0_f(\theta)}{\partial x} \\ \frac{\partial L^1_f(\theta)}{\partial x} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.20}$$

Clearly the observability is a unity matrix and has a rank of 4, so the 2-D DaNI system is locally observable everywhere.

4.2.2 Design of sliding-mode controller

With controllability and observability verified, a SMC controller can be established using the dynamic equations, measurements, and inputs defined above. A SMC controller works this way: first draw the error dynamics to be near the zero-error-surface, then chatters around the zero-error-surface, and finally to the origin.

The relative order has to be evaluated first to determine whether zero-dynamics has to be included in the analysis. Relative order for a single output is defined as the highest order of derivative of output over time when the input first shows up explicitly in the expression. For example, for a output $h(x)$, its derivative over time is,

$$\frac{\partial h}{\partial t} = \frac{\partial h}{\partial x} \cdot \dot{x} = \frac{\partial h}{\partial t} \cdot (f + g \cdot u) = L_f^1(h) + L_g^1(h) \cdot u = L_f^1(h), \text{ if } L_g^1(h) = 0. \tag{4.21}$$

Then keep calculating the higher derivatives of h over t until at some point $L_g^{i-1}(L_f^{i-1}(h)) \neq 0$. This i is the order for this specific single output. Recall that the output of this DaNI 2-D model has two outputs, expressed,

$$h(x) = \begin{pmatrix} r \\ \theta \end{pmatrix} = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}. \tag{4.22}$$

For the first output $h_1 = r$,

$$\begin{aligned} h_1 = r = L_f^0(h_1) &= \xi_1 \\ \dot{h_1} = L_f^1(h_1) = \frac{\partial h_1}{\partial x} \cdot f &= v_r = \xi_2 \\ \text{since } L_g^1(L_f^1(h_1)) = \frac{\partial v_r}{\partial x} \cdot g &= \begin{pmatrix} g_{21} & g_{22} \end{pmatrix} \neq 0 \\ \ddot{h_1} = L_f^2(h_1) + L_g^1(L_f^1(h_1)) \cdot u = \frac{\partial v_r}{\partial x} \cdot f + \begin{pmatrix} g_{21} & g_{22} \end{pmatrix} u &= C_r + D_r \cdot u \end{aligned} \tag{4.23}$$

For the second output $h_2 = \theta$,

$$\begin{aligned} h_2 = \theta = L_f^0(h_2) &= \xi_3 \\ \dot{h_2} = L_f^1(h_2) = \frac{\partial h_2}{\partial x} \cdot f &= \omega = \xi_4 \\ \text{since } L_g^1(L_f^1(h_2)) = \frac{\partial \omega}{\partial x} \cdot g &= \begin{pmatrix} g_{41} & g_{42} \end{pmatrix} \neq 0 \\ \ddot{h_2} = L_f^2(h_2) + L_g^1(L_f^1(h_2)) \cdot u = \frac{\partial \omega}{\partial x} \cdot f + \begin{pmatrix} g_{41} & g_{42} \end{pmatrix} u &= C_\theta + D_\theta \cdot u \end{aligned} \tag{4.24}$$

As a result, total relative order is 2+2 = 4, which equals the order of the system. Thus there

is no zero dynamics involved.

The SMC controller does its job by comparing the reference output and its higher-order dynamics to those of real outputs. Thus the normal form, which contains not only real outputs but their high-order derivatives, has to be established as,

$$\xi = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{pmatrix} = \begin{pmatrix} h_1 \\ \dot{h_1} \\ h_2 \\ \dot{h_2} \end{pmatrix}. \tag{4.25}$$

Next, a sliding surface needs to be defined. Two sliding surfaces are defined for both $h_1 = r$ and $h_2 = \theta$, i.e.,

$$\begin{aligned} S_r = \Lambda_r \cdot \Delta_r = (\xi_2 - \dot{r_d}) + \lambda_1 \cdot (\xi_1 - r_d) \\ S_\theta = \Lambda_\theta \cdot \Delta_\theta = (\xi_4 - \dot{\theta_d}) + \lambda_3 \cdot (\xi_3 - \theta_d) \\ S = \begin{pmatrix} S_r \\ S_\theta \end{pmatrix}. \end{aligned} \tag{4.26}$$

In the time domain, out target is to regulate $\Delta \to 0$ so that the system outputs follow the desired outputs. In a SMC control, system output error first goes to the zero-surface ($S = 0$) and then moves to the origin ($\Delta = 0$) with chattering.

Assume now the system satisfies $S = 0$, Λ has to be stable so that $S = 0$ implies eventually $\Delta = 0$. A simple approach to select the surface coefficients is demonstrated below,

$$S_r = \Lambda_r \cdot \Delta_r = \dot{\delta r} + \lambda_1 \cdot \delta r = 0 \to \dot{\delta r} = -\lambda_1 \cdot \delta r. \tag{4.27}$$

A position λ_1 should be selected to ensure stability of the dynamic equation in Eq.4.27. However, Eq.4.27 does not reveal any information about the dynamics of $\ddot{\delta r}$. A preferable approach to determine the surface coefficients is to make not only $\dot{\delta r} \to 0$ and $\delta r \to 0$ but $\ddot{\delta r} \to 0$. Using the state-space system, surface coefficients can be determined by making the

closed-loop second-order system stable. That is,

$$\begin{pmatrix} \dot{\delta} \\ \ddot{\delta} \end{pmatrix} = Ax + Bu = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta \\ \dot{\delta} \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta \\ \dot{\delta} \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} k_1 & k_2 \end{pmatrix} \begin{pmatrix} \delta \\ \dot{\delta} \end{pmatrix}$$
$$\begin{pmatrix} \dot{\delta} \\ \ddot{\delta} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -k_1 & -k_2 \end{pmatrix} \begin{pmatrix} \delta \\ \dot{\delta} \end{pmatrix}. \tag{4.28}$$

Poles of this closed-loop system can be selected so it is at least stable up to its second order dynamics $\ddot{\delta}$. In our case, poles for the SMC controller are $[-2+1j, -2-1j]$. Surface coefficients can then be calculated by,

$$\Lambda_r = \begin{pmatrix} k_1 & k_2 \end{pmatrix} / k_2. \tag{4.29}$$

A SMC controller consists of two portions: equivalent control and sliding control. First of all, a sliding control portion is required to attract the zero dynamics to the zero surface ($S = 0$). There are many types of sliding control functions, including $sign()$, $tanh()$, $hardlimit()$, and etc. The $sign()$ function has been proven to be more robust than the rest. Thus, a sliding control applying $sign()$ function is established as,

$$\begin{pmatrix} L_{g_1}(L_f(r)) & L_{g_2}(L_f(r)) \\ L_{g_1}(L_f(\theta)) & L_{g_2}(L_f(\theta)) \end{pmatrix} \cdot u_s = \begin{pmatrix} -\eta_1 sign(S_r) \\ -\eta_2 sign(S_\theta) \end{pmatrix}. \tag{4.30}$$

Second, we have to make sure that the error dynamics can stay on the zero-surface and eventually go to the origin. This is achieved by the first portion of a SMC control: equivalent control u_{eq}. An expression for u_{eq} is obtained from the constraint $\dot{S} = 0$, or,

$$\dot{S} = 0 = \begin{pmatrix} L_{g_1}(L_f(r)) & L_{g_2}(L_f(r)) \\ L_{g_1}(L_f(\theta)) & L_{g_2}(L_f(\theta)) \end{pmatrix} \cdot u_{eq} + \begin{pmatrix} L_f^2(r) \\ L_f^2(\theta) \end{pmatrix} - \begin{pmatrix} \ddot{r_d} \\ \ddot{\theta_d} \end{pmatrix} + \begin{pmatrix} \lambda_1(\xi_2 - \dot{r_d}) \\ \lambda_3(\xi_4 - \dot{\theta_d}) \end{pmatrix}. \tag{4.31}$$

Finally, the SMC controller is the combination of equivalent control and sliding control,

$$u = u_{eq} + u_s. \tag{4.32}$$

4.2.3 Pre-filter

In practical applications, very often only system outputs are available, which are r and θ in this DaNI trajectory control system. As shown above, the SMC controller for this 2-D DaNI system requires the second and third derivatives of the desired outputs. Higher-order derivatives can be estimated using a tracker with the third-order system,

$$\begin{pmatrix} \dddot{y}_{df} \\ \ddot{y}_{df} \\ \dot{y}_{df} \end{pmatrix} = (A_{df} - B_{df} \cdot K_{filter}) \begin{pmatrix} \ddot{y}_{df} \\ \dot{y}_{df} \\ y_{df} \end{pmatrix} + B_{df} \left(C_{df} (B_{df} \cdot K_{filter} - A_{df})^{-1} B_{df} \right)^{-1} y_d$$
$$A_{df} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \; B_{df} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \; C_{df} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}. \tag{4.33}$$

The first part of Eq.4.33 ensures that y_{df} converges to 0, and the second part ensures that y_{df} actually follows non-zero y_d. Since both r and θ has a relative order of 2, they share the same form for pre-filter as in Eq.4.33. The value of K_{filter} is still obtained from pole placement. Response of the pre-filter must be sufficiently faster than that of the SMC controller, so the poles for the pre-filter are chosen as, $[-2 + 1j, -2 - 1j, -2] \cdot 15$.

4.2.4 Extended luenberger observer

Since the measurements are only r and θ, the rest of the states have to be rebuilt from the available measurements. As proven, this 2-D DaNI trajectory control system is locally observable everywhere. Thus, an extended Luenberger observer can be applied. The

observed system and real system, as well as their linear approximations, are,

$$\dot{\hat{x}} = f(\hat{x}) + g(\hat{x})u + M(y - \hat{y}) \rightarrow \dot{\hat{x}} = \frac{\partial f}{\partial x}|_{\hat{x}} + \frac{\partial (gu)}{\partial u}|_{\hat{x}} u + MC(x - \hat{x}) = A_o \hat{x} + B_o u + MC_m(x - \hat{x})$$

$$\dot{x} = f(x) + g(x)u \rightarrow \dot{x} = \frac{\partial f}{\partial x}|_{x} + \frac{\partial (gu)}{\partial u}|_{x} u = A_o x + B_o u$$

$$C_m = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.34}$$

To determine M, the error dynamics has to be stable by selecting proper poles for,

$$\dot{e} = \dot{\hat{x}} - \dot{x} = A_o e - MC_m e, \;\; M = (\text{Place}(A_o^T, C_m^T, \text{poles}))^T. \tag{4.35}$$

Speed of the observer must be sufficently faster than that of the SMC controller, thus poles for the observer are chosen as $[-2+1j, -2-1j, -3+1j, -3-1j] \cdot 30$. As seen here, M is changing with time if the system is nonlinear. Placing ploes is a computationally expensive command and will require much more time if varying M during simulation. In practice, a constant M that works for the system range is generated and used during the whole simulation. To simulate uncertainties and errors, measurement error and disturbances are applied only to the system equations for real system, while the observed system is assumed to be free of uncertainy or error. To simulate parameter uncertainties, slightly different model parameters are used for the observed system.

4.3 Simulation of SMC controller

To examine the SMC trajectory controller developed above, simulations are conducted in MATLAB, and results visualized by MATLAB plots.

4.3.1 Simulation workflow

After initialization, the simulation generates a desired trajectory in time in the global frame for the DaNI. Alternatively, the user can be prompted to draw a random trajectory. Note that hand-drawn trajectory may have high-frequency noise that introduces a much higher requirement on the controller. Therefore, hand-drawn option is only for demonstration of the code, but discarded in simulation results analysis. Next, an ordinary differential equation solver is applied to calculate simulated state history of the 2-D DaNI system. Lastly, relevant results are plotted to present the performance of the SMC controller. Each step will be described below.

Initialization defines six "struct"-type variables: $para, sys, elo, SMC, pf$, and dis. The variable $para$ specifies simulation parameters: start time, end time, and time step. A time step of 0.02 s is selected to avoid high-frequence change of modulations so that the control input varying frequency does not surpass the capability of the motor controller on DaNI. Also the time step cannot be too large, otherwise the ode solver will not yield an accurate result. The variable sys (Eq.4.36) contains all the parameters and functions associated with the real system (with accurate parameters, uncertainties, disturbances, and errors).

$$\begin{aligned} &\text{Number of States; Number of Measurements; Order of System} \\ &f_x@x;\ g_x\ const \\ &h_m : @x \\ &\text{Right Driving Force } F_{rdrivex}@(m_{crr}, vr, w) \\ &\text{Left Driving Force } F_{ldrivex}@(m_{cll}, vr, w) \\ &\text{Right Motor Current } I_r@(m_{crr}, vr, w) \\ &\text{Left Motor Current } I_l@(m_{cll}, vr, w) \\ &\text{Initial State } x_0 \end{aligned} \tag{4.36}$$

Here, f_x, g_x, jointly define system dynamics, and h_m stands for available measurements. The variable elo is a structure defined similar to sys, as shown in Eq.4.37, with inaccurate

parameters but without any disturbance.

$$\begin{aligned} \text{Whetherincludingobserver} &: \text{isELO} \\ &\text{zetaTrans} \\ \hat{fx}@\hat{x};\ \hat{g_x}\ &const \\ \hat{h}_m@\hat{x} & \\ C_r;\ D_r & \\ \text{Linearized Matrix A}\ \ \hat{A}_x@\hat{x} & \\ \text{Linearized Matrix C}\ \ \hat{C}_x@\hat{x} & \\ \text{Observed Initial State}\ \hat{x}_0 & \end{aligned} \tag{4.37}$$

Whether the observer is included in the simulation or not is controlled by the variable isELO. $zetaTrans : \hat{x} \to \xi$ is a mapping function that transforms observed states to norm form as defined in the SMC controller section. The variables Cr, Dr are SMC elements used to construct the SMC controller, and $\hat{A}_x, \hat{C}_x$ are used for the obsever design. Other variables are defined the same way as in sys, but all as functions of observed state $\hat{x}$. SMC structure includes A, B martix which determine sliding surface coefficients, poles for pole placement, and sliding coefficient (η in $-\eta \cdot sign(S)$). pf collects all parameters related to pre-filter design, including those matrices defined in pre-filter section, selected poles for pole placement, and pre-filter initial states. dis determines whethere a certain type of error or noise is present in the system, such as parameter errors in the observed system and noises in the real system.

After initialization, the pre-filter calculates higher-order dynamics of the desired trajectory. Lastly, ODE solver takes over the simulation along with pre-defined system, observer, and SMC models. Control inputs obtained from the SMC strategy is modified to low-resolution values to reflect the real control input resolution on the DaNI. As discussed in Chapter 3, the encoders on DaNI only accept nominal commands such as -0.2, 0.3, 0.8, and will automatically modify commands such as -0.24, 0.47 to their nearest one-digit nominal values. Since the route simulated here is much more accurate than what DaNI could follow, modulations are modified to their nearest two-digit nominal values before being applied to

dynamic equations. In MATLAB, this is done by the command,

$$\text{modified input} = round\ (\text{input} \cdot 100)/100. \tag{4.38}$$

In addition, all calcuated control inputs with a magnitude larger than 1 is restricted to ±1.

4.3.2 Simulation results

First, assume there is no input resolution restriction and no observer or disturbance. This means the pulse-width modulation could assume any value in range with any level of resolution. Simulation results are shown in Figure 4.3. The DaNI is expected to accelerate in a straight line with a constant acceleration for a certain period, then a uniform motion for a short period. Next the DaNI makes a left turn for a quarter circle with the same speed and a constant angular velocity. Lastly it makes a right turn for another quarter circle with the same speed and angular velocity (reverse direction). The upper right plot compares the desired, filtered, and resulting cumulative trajectory length and yaw angle. The resulting curves under SMC control follow the desired curves very well. The middle left figure shows that left and right modulations increase simutaneously during straight constant acceleration and remain constants during uniform motion. As expected, right motor modulation surpasses the left during a left turn and vice versa. In the middle right figure, it is clear that the speed of DaNI does not exceed its limit. Yaw rate is zero during straight motion and a constant value during uniform turnings. The lower left figure depicts variation in driving forces at both wheels. At the start, both driving forces increase to a high value in a short time to accelerate the DaNI and remains at that value during the constant acceleration. Then during uniform motion, no matter if straight or rotational, driving forces at the two wheels are rather low. This is because during those phases only motor/drivetrain resistance and rolling resistance have to be overcome. At turnning points where the DaNI is trying to alter its orientation, a distinct difference in driving forces at the two wheels are observed. Motor

current history can be measured directly and reveals similar information as the driving force, but is less intuitive since currents do not affect the DaNI motion in as direct a way as the driving forces do.

Second, assume there is input resolution restriction, but still no observer or noise. Simulation results are shown in Figure 4.4. As compared to the previous case where no input resolution restriction is present, the actual trajectory length deviates a little from the desired one. There are evident chatters in the modulation plot. At a certained point, calculated input modulation may have a higher resolution than the restricting limit and has been modified to a different value with lower resolution.

Finally, assume there is input resolution restriction, observer, and noise. Simulation results are shown in Figure 4.5. Obviously noise is present in the history of modulation, rates, current, and forces. However, the actual cumulative trajectory length and yaw angle follow the desired ones well, thus demonstrating the robustness of the sliding-mode controller.

Theoretically speaking, a serial resistance R_b can be included in the 2-D DaNI model to restrict the power limit. The only change of the system equations is to replace the battery supply voltage V_b to a revised value V_{in}, i.e.,

$$\begin{aligned} i_m &= \frac{V_{in} - r_m \omega_m}{R_m} \\ V_{in} &= mc \cdot (V_b - R_b i_b) \\ i_b &= mc \cdot i_m \\ V_{in} &= \frac{mcV_b + \frac{mc^2 r_m \omega_m R_b}{R_m}}{1 + mc^2 \frac{R_b}{R_m}}. \end{aligned} \tag{4.39}$$

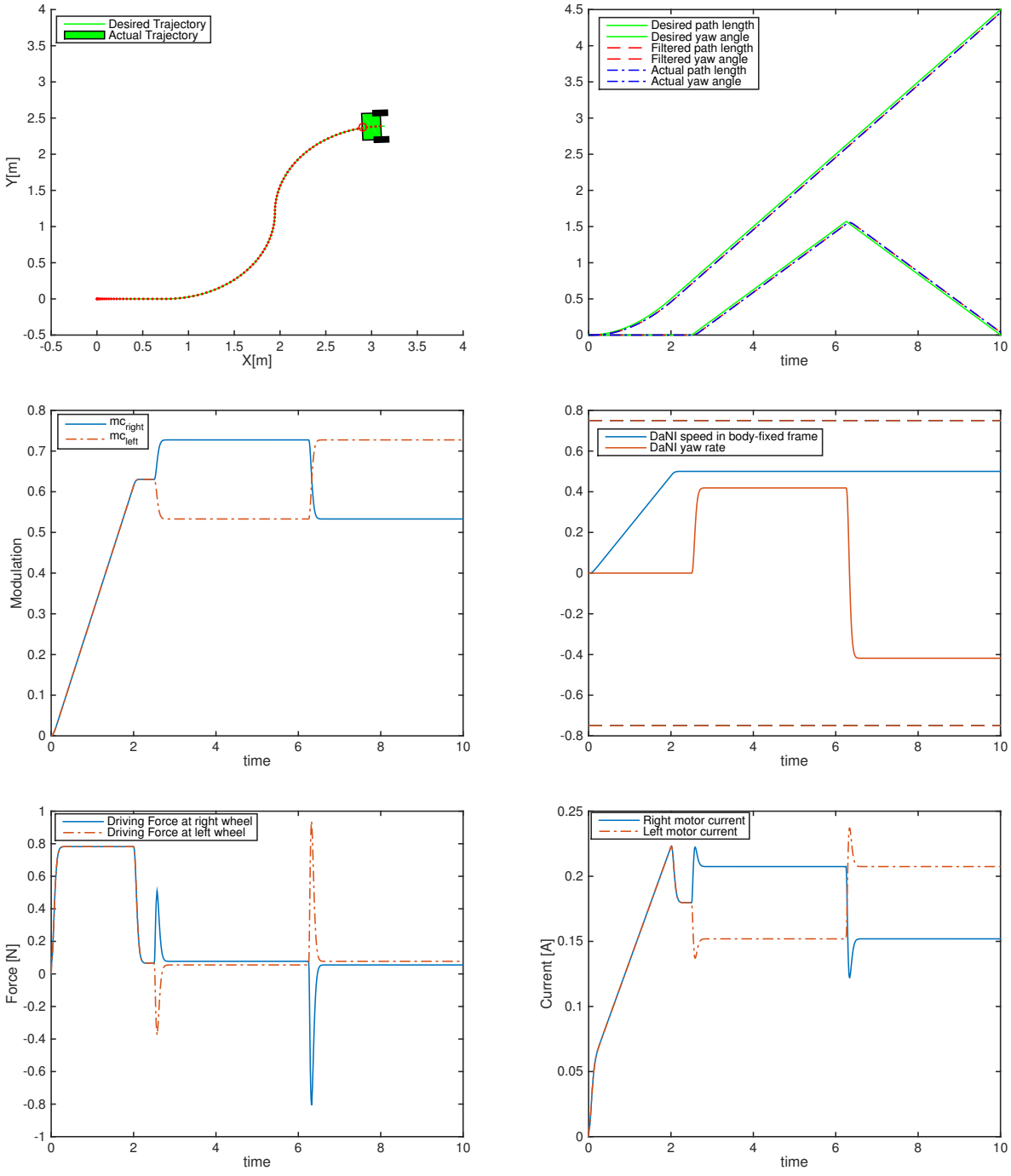

Figure 4.3: Simulation with no modulation resolution restriction, observer, error, or noise

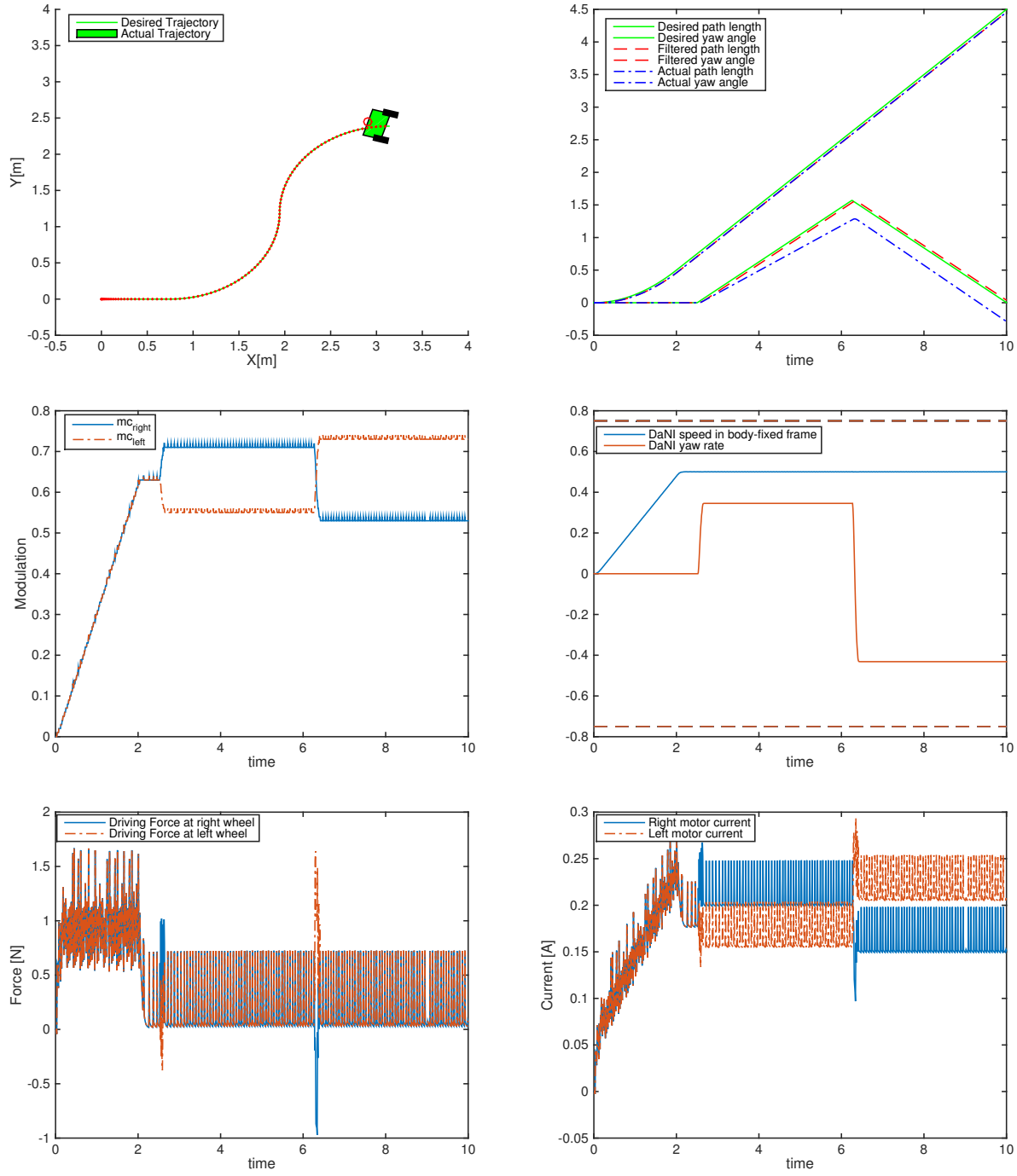

Figure 4.4: Simulation with modulation resolution restriction, but without observer, error, or noise

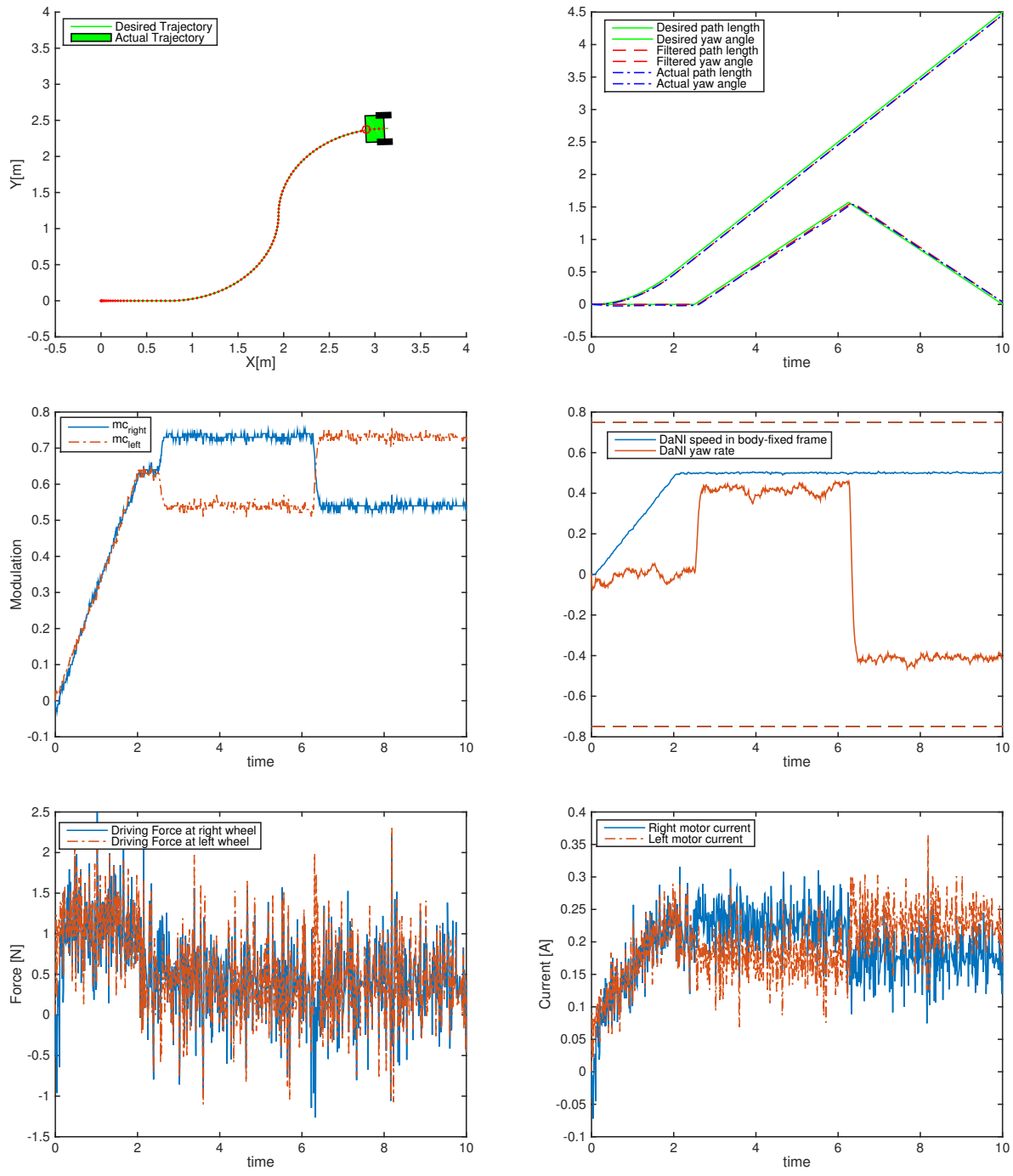

Figure 4.5: Simulation with modulation resolution restriction, observer, error, and noise

If replacing V_b with V_{in}, partial system dynamics become,

$$
\begin{aligned}
F_{xl} &= \frac{GR}{R_w}\Big(\frac{\frac{m_{cl}V_b + \frac{m_{cl}^2 r_m \omega_m R_b}{R_m}}{1+m_{cl}^2\frac{R_b}{R_m}} r_m}{R_m} m_{cl} - \big(\frac{r_m^2}{R_m} + B_r\big)\big(v_r - \omega\frac{B}{2}\big)\frac{GR}{R_w}\Big) \\
F_{xr} &= \frac{GR}{R_w}\Big(\frac{\frac{m_{cr}V_b + \frac{m_{cr}^2 r_m \omega_m R_b}{R_m}}{1+m_{cr}^2\frac{R_b}{R_m}} r_m}{R_m} m_{cr} - \big(\frac{r_m^2}{R_m} + B_r\big)\big(v_r + \omega\frac{B}{2}\big)\frac{GR}{R_w}\Big) \\
\dot{v} &= \frac{F_{xr} - F_{rr} + F_{xl} - F_{rl}}{m_{eff}}.
\end{aligned}
\tag{4.40}
$$

Now $\dot{v}$ is not linearly proportional to control input $u = [m_{cr}, m_{cl}]^T$. Therefore, if adding the serial resistance, the entire system dynamics can no longer be modeled in an affine form $\dot{x} = f(x) + g(x)u$. In practice, neglecting the serial resistance won't be a problem as long as the speed of the trajectory is limited and the resulting motor current does not exceed limit.

4.4 Impact of continuous control on model-based assertions

The performance of the continuous trajectory controller not only affects the accuracy of controlled vehicle trajectory, but also impacts the design of model-based assertions for the AIMS. When designing a model-based assertion, there is always an event that will trigger an assertion failure. One primary model-based assertion regarding the accuracy of the actual vehicle trajectory would be the maximum window of possible deviation from the desired trajectory. The associated trigger event will be a vehicle going out of the maximum deviation window. If the controller says that the deviation window is expected to be δ, then the assertion failure is triggered for a deviation of 2δ. A small δ enables the AIMS to operate with a higher accuracy without an assertion failure. On the other hand, a large δ will force the cyber-physical system to partially lose its assurance of detailed vehicle status.

For example, the open-loop controller and SMC controller are applied to separate vehicles within the AIMS. Both Gaussian noise and disturbance are corrupting the vehicle

dynamics. A constant disturbance might be handled by an open-loop controller with designated modification in inputs, while a random disturbance and Gaussian noise cannot be well-compensated. Thus given both Gaussian noise and random disturbance, the expected deviation in vehicle trajectory under an open-loop control can be very large. The trajectory deviation assertion designed accordingly would be much less informative for the AIMS. However, for bounded disturbances and noise, a SMC controller is able to, if the control effort is large enough, keep the vehicle on the desired trajectory with acceptable deviations. The control coefficient can be analytically selected beforehand based on the known bounds of disturbance and noise. Therefore, a good continuous trajectory controller offers more accurate information to the AIMS so that a more informative and helpful model-based assertion can be designed.

Chapter 5: Conclusion

A two-player game has been formulated for both over-approximated and concrete traffic networks. It takes much less computational time for TuLiP to synthesize a controller for the over-approximated system than the concrete system. Therefore conducting a preliminary check on the infeasibility of traffic specifications before working on the concrete traffic network could save a lot time in the early design stage of a traffic network. In the simulation, GR(1) specifications qualified in the over-approximated traffic network is verified to be realizable in the concrete network. However, as the number of vehicles and complexity of traffic network increase the computational time for synthesizing a discrete planner using TuLiP becomes rather computationally expensive. In furture work, either a more efficient way to model the traffic network or a more efficient synthesis tool should be pursued.

A complete AIMS was developed in LabVIEW with DaNI robot vehicles. This laboratory system features typical characteristics of a cyber-physical system, such as vehicle-manager communication, system-level allocation of intersection traffic, vehicle-level trajectory control, and etc. Functionality of model-based assertions are demonstrated and verified in this LabVIEW AIMS. Only two robot vehicles are used in this implementation for the sake of research testing. In future research, more robot vehicles with a more complex traffic network could be used and thus a larger experimental space required. An open-loop control is adapted in this implementation because of a lack in accurate feedback position measurement. For one robot vehicle, position tracking is feasible using a vision capturing system developed

for this research. For motion capture of multiple robot vehicles, either an advanced sensor fusion along with IMU and vision capturing system, or a high-speed camera tracking system, should be introduced. With such position measurements available, any feedback closed-loop controller could be implemented in this LabVIEW AIMS.

To improve the robot vehicle trajectory accuracy, a sliding-mode controller with extended observer and pre-filter is developed and simulated. It turns out that SMC controller could offer an exceptional control performance, even in the presence of input resolution constraint, model parameter error, measurement noise, and system disturbance. However, slipping, a significant phenomenon for any vehicle dynamics, is not included in the 2-D vehicle dynamics model used in this research. In future work, slipping should be taken into consideration. Also, SMC controller may be implemented on a laboratory robot vehicle once multi-position tracking is available.

For a typical cyber-physical system design, a system-level planner requires a discrete control synthesis strategy to coordinate all physical components within the system. Then each physical component receives discrete commands and implements them through low-level continuous controls. System-level and component-level model-based assertions can then be integrated into the cyber-physical system to further enhance safety and liveness.

Appendix A: Code for the laboratory AIMS

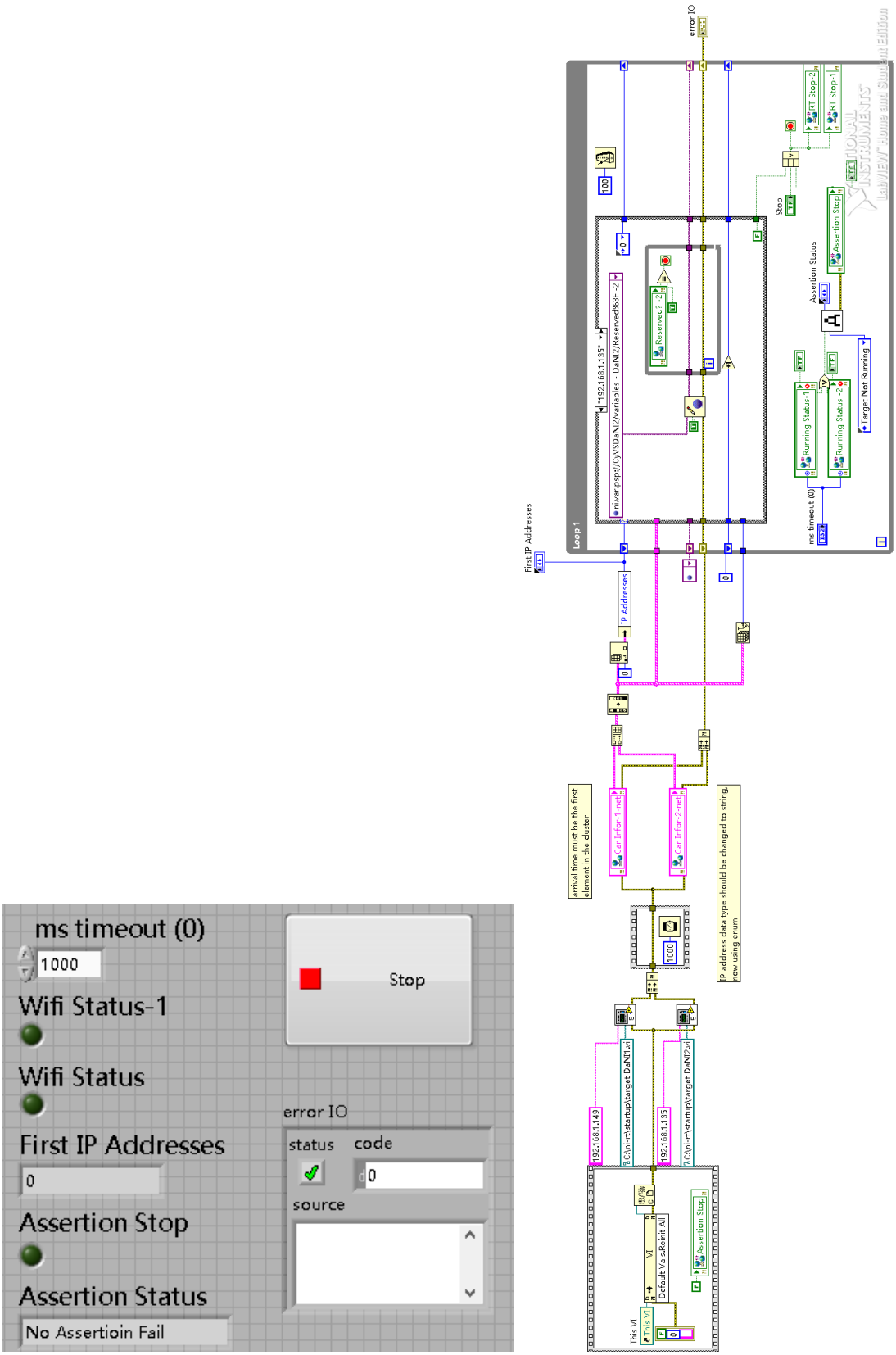

Figure 6.1: Intersection Manager LabVIEW front panel and block diagram

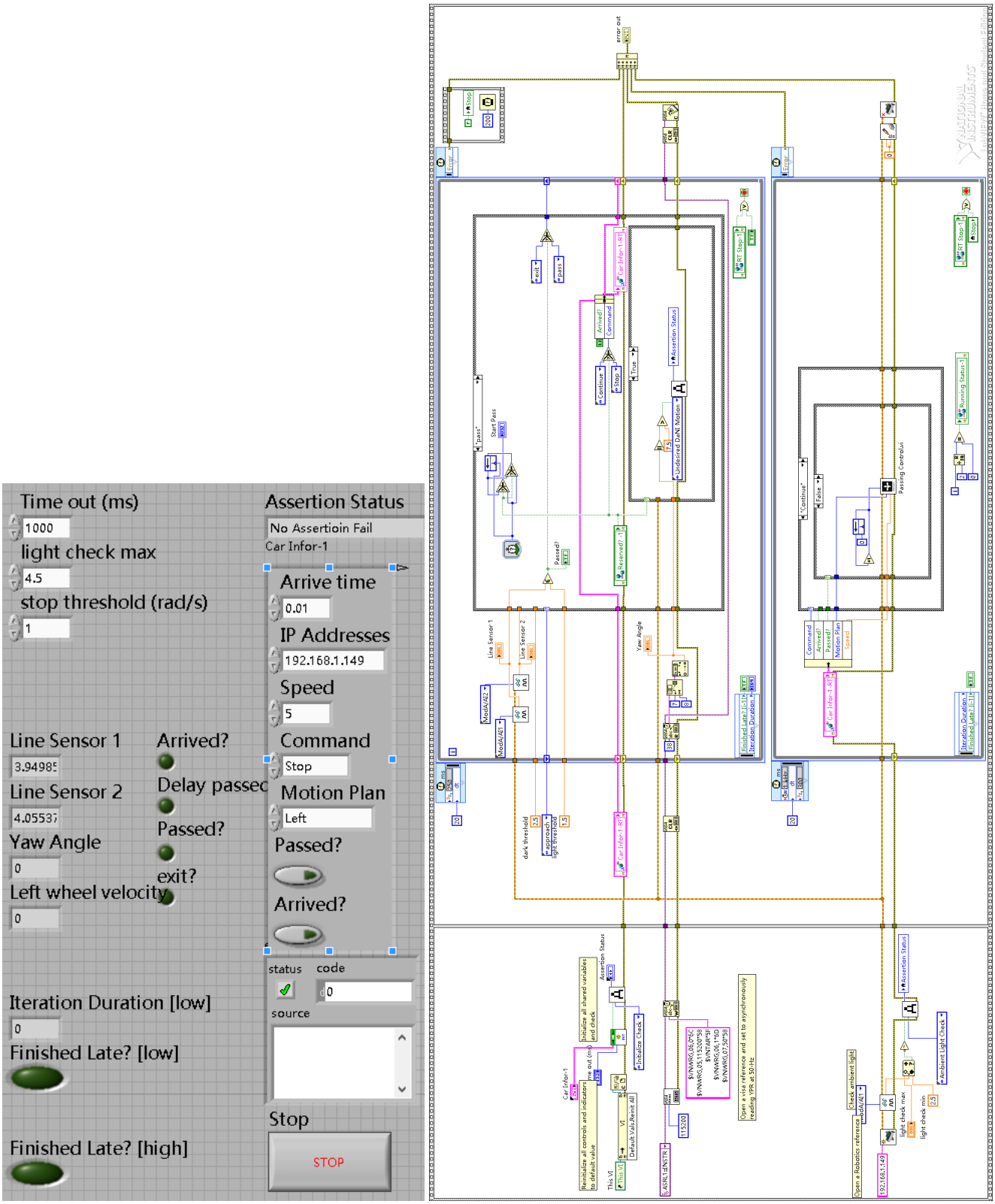

Figure 6.2: DaNI LabVIEW front panel and block diagram

Appendix B: Code for continuous trajectory control

Main

```
%   The University of Texas at Austin
%   Mechanical Engineering Department
%   Dongyi Zhou
%   Main.m

clear all
clc
close all

% Define global variables
global para dis elo sys  SMC pf PFfun   % system struct
global xbody thetabody
is = 4.55; % stall current, A (from specs)

%%%%%%%% Adjustable parameters
StartTime= 0;
para.EndTime= 10;
para.h = 0.01;  %step size for RK4, should be <0.01
para.TimeVec=StartTime:para.h:para.EndTime;
para.axisrange = [-0.5 4 -0.5 4];
%Draw or Animation?
Draw = false;
Animation = false;
%observer?
elo.isELO = false;
```

```
%system disturbance defination
dis.HasSysUncertainty=elo.isELO;     %whether add system uncertainty
dis.HasStateNoise=elo.isELO;          %whether add noise into states equation
dis.HasMeasurementNoise= elo.isELO;    %whether measurement noise exists

%%%%%%%%% Define the path and calculate corresponding [x,theta] in body-fixed frame
if (~Draw)
[Xglobal,Yglobal] = generatepath(para);
else
    h1=figure;
    axis(para.axisrange);
    xlabel('X');
    ylabel('Y');
    title('Draw a SMOOTH trajectory SLOWLY'); %if draw faster, the DaNI has to move faster
    DrawPoints = get_pencil_curve(h1);
    Xglobal = DrawPoints(:,1);
    Yglobal = DrawPoints(:,2);
    diffXY = [diff(Xglobal),diff(Yglobal)];
    index = [];
    for i = 1:size(diffXY,1)
        if isequal(diffXY(i,:),[0,0])
        index = [index,i];
        end
    end
    Xglobal(index)=[];
    Yglobal(index)=[];
    disp('Finish drawing, Starting calculation');
    Xglobal = interp1(linspace(0,para.EndTime,length(Xglobal)),Xglobal,para.TimeVec,...
'linear','extrap');
    Yglobal = interp1(linspace(0,para.EndTime,length(Yglobal)),Yglobal,para.TimeVec,...
'linear','extrap');
end

% at this point, Xglobal & Yglobal already has the same size as of
% para.TimeVec
[xbody,thetabody] = global2body(Xglobal,Yglobal);
fprintf('%d points collected \n',length(Xglobal));
fprintf('Is the xbody array monotonically increasting? : %d\n', all(diff(xbody)>0));
```

```
%%%%%%%%% real system
sys.NumOfStates=4;
sys.NumOfMeas  =2;
[~,fx,gx,~,hmx,~,~,~,~,order,Frdrivex,Fldrivex,RI,LI]=CalNormForm(...
dis.HasSysUncertainty,false);
sys.fx = fx;
sys.gx = gx;
sys.hm = hmx;
sys.order = order;
sys.Frdrivex = Frdrivex;
sys.Fldrivex = Fldrivex;
sys.RightCurrent = RI;
sys.LeftCurrent  = LI;
% state order:  r vr theta w
Po = Map(0);
sys.x0=[Po(1),0,Po(2),0]';

%%%%%%%%% observer(modeled system)
[zetaTrans,fx,gx,~,hmx,Cr,Dr,Ax,Cx,~,~,~,~,~]=CalNormForm(...
dis.HasSysUncertainty,elo.isELO);
elo.zetaTrans = zetaTrans;
elo.fx = fx;
elo.gx = gx;
elo.hm = hmx;
elo.Cr = Cr;
elo.Dr = Dr;
elo.Ax = Ax;
elo.Cx = Cx;
elo.x0 = sys.x0+[0,0,0.01,0]';  % assume very small start error

%%%%%%%% Prefilter for r and theta has similar structure (order(r)=order(theta)
%= sys.order/2, otherwise more detains needs to be modified.
% Prefilter structure: y is estimated states, r(t) is reference
% y_dot=(Ayr-Byr*Kyr)y+Byr*Pyr*r(t)
pf.size = sys.order/2+1;
pf.Ayr = [[zeros(pf.size-1,1),diag(ones(1,pf.size-1))];zeros(1,pf.size)];
pf.Byr=[zeros(pf.size-1,1);1];
pf.Cyr=[1,zeros(1,pf.size-1)];
pf.Poleyr=[-2+1j -2-1j -2]*15;   %pole selected
```

```
pf.Kyr=place(pf.Ayr,pf.Byr,pf.Poleyr);                %pole placement
tmp=pf.Cyr*inv(pf.Byr*pf.Kyr-pf.Ayr)*pf.Byr;
pf.Pyr=inv(tmp);                              %constant Pyr
Po = Map(0);
pf.rT0=[Po(1), zeros(1,pf.size-1)];
pf.thetaT0=[Po(2),zeros(1,pf.size-1)];

%%%%%%%%% Design sliding surface for SMC
SMC.size = sys.order/2;
SMC.As = [[zeros(SMC.size-1,1),diag(ones(1,SMC.size-1))];zeros(1,SMC.size)];
SMC.Bs=[zeros(SMC.size-1,1);1];
SMC.PoleSMC=[-2+1j,-2-1j];
KsSMC=place(SMC.As,SMC.Bs,SMC.PoleSMC);
SMC.Ks=KsSMC/KsSMC(length(KsSMC));
SMC.ita = 0.1;  %???????? originally 5

fprintf('start calculation');
tic
[States,Input]=Controller();
% Input = Input;
toc

%%%%%%%%%%
fprintf('output states contain NaN? : %d\n', any(num2str(reshape(States,1,...
numel(States)))=='N'));
fprintf('control input contain NaN? : %d\n', any(num2str(reshape(Input,1,numel(Input)))...
=='N'));
%%%%%%%% get global (X,Y) from body-fixed coordinates (r,theta)
%since length of path is harmonic (increasting)
Xreal = interp1(xbody,Xglobal,States(:,1),'linear','extrap');
Yreal = interp1(xbody,Yglobal,States(:,1),'linear','extrap');
fprintf('Xreal contain any NaN? : %d\n', any(num2str(reshape(Xreal,1,numel(Xreal)))...
=='N'));
fprintf('Yreal contain any NaN? : %d\n', any(num2str(reshape(Yreal,1,numel(Yreal)))...
=='N'));

%%%%%%%% plot
fig1 = figure (1);
set(fig1, 'Position', [ 20   600   400   300]);%[x,y,width,height]
```

```
axis(para.axisrange);
hold on
xlabel('X[m]');
ylabel('Y[m]');

if ~Animation
    plot(Xglobal, Yglobal,'g-',Xreal,Yreal,'r-.','LineWidth',0.5);
else
    plot(Xglobal, Yglobal,'g-','LineWidth',1);

    [xb, yb, xrc, yrc, xflw, yflw, xfrw, yfrw] = DaNI(Xreal(1),Yreal(1),States(1,3));
    %Define component plots
    plotzb = fill(xb, yb,'g');  % Plot robot base
    plotzrc = plot(xrc, yrc, 'ro');  % Plot rear pivot
    plotzflw = plot(xflw, yflw, 'k','linewidth',5);  % Plot rear left wheel
    plotzfrw = plot(xfrw, yfrw, 'k','linewidth',5);  % Plot rear right wheel

    for k = 1:10:length(Xreal)
        plot(Xreal(k),Yreal(k),'r.');

        [xb, yb, xrc, yrc, xflw, yflw, xfrw, yfrw] = DaNI(Xreal(k),Yreal(k),States(k,3));
        % Plot vehicle - updates data in each plot
        set(plotzb,'xdata',xb);
        set(plotzb,'ydata',yb);
        set(plotzrc,'xdata',xrc);
        set(plotzrc,'ydata',yrc);
        set(plotzflw,'xdata',xflw);
        set(plotzflw,'ydata',yflw);
        set(plotzfrw,'xdata',xfrw);
        set(plotzfrw,'ydata',yfrw);
        pause(0.2);
    end
end
legend('Desired Trajectory','Actual Trajectory');
hold off

fig2 = figure (2);
set(fig2, 'Position', [ 450   600   400   300]);
plot(para.TimeVec,Input(:,1),para.TimeVec,Input(:,2),'-.');
```

```
xlabel('time');
ylabel('Modulation');
legend('mc_{right}','mc_{left}');

% plot vr and omega
fig3 = figure(3);
set(fig3, 'Position', [900,600,400,300]);
plot(para.TimeVec,States(:,2),para.TimeVec, States(:,4),para.TimeVec,0.75*ones(...
length(para.TimeVec)),'--',para.TimeVec,-0.75*ones(length(para.TimeVec)),'--');
xlabel('time');
legend('DaNI speed in body-fixed frame','DaNI yaw rate');

% plot Fxr and Fxl
for i = 1:length(para.TimeVec)
    Frdrive(i) = sys.Frdrivex(Input(i,1),States(i,2),States(i,4));
    Fldrive(i) = sys.Fldrivex(Input(i,2),States(i,2),States(i,4));
end

fig4 = figure(4);
set(fig4, 'Position', [20,50,400,300]);
plot(para.TimeVec,Frdrive,para.TimeVec,Fldrive,'-.');
xlabel('time');
ylabel('Force [N]');
legend('Driving Force at right wheel','Driving Force at left wheel');

%cal and plot current
for i = 1:length(para.TimeVec)
    RightCurrent(i) = sys.RightCurrent(Input(i,1),States(i,2),States(i,4));
    LeftCurrent(i) = sys.LeftCurrent(Input(i,2),States(i,2),States(i,4));
end
fig5 = figure(5);
set(fig5, 'Position', [450,50,400,300])
plot(para.TimeVec,RightCurrent,para.TimeVec,LeftCurrent,'-.');
xlabel('time');
ylabel('Current [A]');
legend('Right motor current','Left motor current');

% plot desired, filtered, and actual states in body-fixed frame
PF = zeros(length(para.TimeVec),6);
```

```
for i = 1:length(para.TimeVec)
PF(i,:) = PFfun(para.TimeVec(i))';
end
fig6 = figure(6);
set(fig6, 'Position', [900,50,400,300]);
plot(para.TimeVec,xbody,'g-',para.TimeVec,thetabody,'g-',para.TimeVec,PF(:,1),'r--',...
para.TimeVec,PF(:,4),'r--',para.TimeVec,States(:,1),'b-.',para.TimeVec,States(:,3),'b-.');
xlabel('time');
legend('Desired path length','Desired yaw angle','Filtered path length',...
'Filtered yaw angle','Actual path length','Actual yaw angle');

%end of function 'main'
```

Controller

```
%Controller design and state calculation
% Main function for this page
function [xout,uout]=Controller()
global  sys para elo PFfun
TimeVec=para.TimeVec;
PFfun=Prefilter(TimeVec);
xout=CalState(TimeVec,sys,para,elo);
uout = Calcontrol(TimeVec,xout);
end

% Prefilter for FBL and SMC
function [PFfun]=Prefilter(TimeVec)
global para pf var
var = 'r';
rT0=pf.rT0;
PFr=RK4(TimeVec,rT0,para.h,@DynPF);
var = 'theta';
thetaT0=pf.thetaT0;
PFtheta=RK4(TimeVec,thetaT0,para.h,@DynPF);
PF = [PFr; PFtheta];
PFfun = @(t) (PF(:,1+floor(t/para.h))+(PF(:,1+ceil(t/para.h))-PF(:,1+floor(t/para.h))...
)*(t/para.h-floor(t/para.h)));
end
% Prefilter dynamics
function xdot = DynPF(t,x)
```

```
global  pf var
Ayr=pf.Ayr;  Byr=pf.Byr;    Kyr=pf.Kyr;
Pyr=pf.Pyr;
Aarg=Ayr-Byr*Kyr;
Barg=Byr*Pyr;
Point = Map(t);
switch var
    case 'r'
       Ref = Point(1);
    otherwise
       Ref = Point(2);
end
xdot = Aarg*x+Barg*(Ref);
end

% Calculate Input
function u=Calcontrol(TimeVec,x)
global elo
u = zeros(length(TimeVec),2);
zetaTrans = elo.zetaTrans;
for i=1:length(TimeVec)
zeta=zetaTrans(x(i,5),x(i,6),x(i,7),x(i,8));
u(i,:)=CalInput(TimeVec(i),zeta,x(i,:));
end
end

% Caluclate state changes using RK4
function [xout]=CalState(TimeVec,sys,para,elo)
isELO = elo.isELO;
if (isELO)
    x0= [sys.x0;elo.x0];
else
 x0= [sys.x0;sys.x0];
end
    xout=RK4(TimeVec,x0,para.h,@system);
    xout=xout'; %n*8
end

% close-loop system with controller
```

```
function [xdot]=system(t,x)
global elo
zetaTrans = elo.zetaTrans;
zeta=zetaTrans(x(5),x(6),x(7),x(8));
u=CalInput(t,zeta,x);
xdot=DynEqn(x,u);
end

% Calculate input
function [uout]=CalInput(t,zeta,x)
global  PFfun SMC elo sys

yd=PFfun(t); %[r, vr, vrdot, theta, w, wdot]' (6*n)
Drx = elo.Dr(x(5),x(6),x(7),x(8));
Crx = elo.Cr(x(5),x(6),x(7),x(8));
delta = [yd(1:sys.order/2);yd(sys.order/2+2:end-1)] -zeta; %r vr, theta, w

us=-SMC.ita*[sign(SMC.Ks*delta(1:sys.order/2)); sign(SMC.Ks*delta((...
sys.order/2+1):end))];% 2*1
uout= -Drx \ (us - ([yd(sys.order/2+1);yd(length(yd))]-Crx +...
SMC.Ks(1:length(SMC.Ks)-1)*[delta(2:sys.order/2);delta((sys.order/2+2):end)]));
%modify the input so that it assumes only nominal values
 uout = modifyinput(uout);
end

function modifiedinput = modifyinput(input)
        if input(1)>1
            modifiedinput(1) = 1;
        elseif input(1)<-1
            modifiedinput(1) = -1;
        else
            modifiedinput(1) = round(input(1)*100)/100;
        end

        if input(2)>1
            modifiedinput(2) = 1;
        elseif input(2)<-1
            modifiedinput(2) = -1;
        else
```

```
            modifiedinput(2) = round(input(2)*100)/100;
        end

        modifiedinput = modifiedinput';
end

% Dynamic equation for the system
function [xdot]=DynEqn(x,u)
global sys elo dis
    kk = sys.NumOfStates;
    isELO =elo.isELO;
    fnum=sys.fx(x(1),x(2),x(3),x(4));
    gnum=sys.gx(x(1),x(2),x(3),x(4));
    xdot(1:kk,1)=(fnum)+(gnum)*u;

if (~isELO)
    xdot(kk+1:2*kk,1)= xdot(1:kk,1);
else
   fnum=elo.fx(x(5),x(6),x(7),x(8));
   gnum=elo.gx(x(5),x(6),x(7),x(8));

%    A = elo.Ax(x(5),x(6),x(7),x(8));
%    C = elo.Cx(x(5),x(6),x(7),x(8));
%    polesELO = [-2+1j,-2-1j,-3+1j,-3-1j]*30;
%    MT = place(A',C',polesELO);
%    M =MT'

   M=1000*[ 0.0939    0.0017
            2.0489    0.1057
            0.0272    0.0883
           -0.3167    0.8999];
   currentsysmea = sys.hm(x(1),x(2),x(3),x(4));
   currentelomea = elo.hm(x(5),x(6),x(7),x(8));
   xdot(kk+1:2*kk,1) = (fnum) + (gnum) * u +...
   M*(currentsysmea+CalMeasurementNoise(dis.HasMeasurementNoise,...
currentsysmea)- currentelomea);
end
if(dis.HasStateNoise )
    xdot(1:kk,1)=xdot(1:kk,1)+CalStateNoise(xdot(1:kk,1));
```

```
    if (~isELO)
     xdot(kk+1:2*kk,1)=xdot(1:kk,1);
    end
end

end
% Noise Generator for measurement
function [CalNoise]=CalMeasurementNoise(isdis,currentsysmea)
global sys
if isdis
% two measurement
    CalNoise=randn(sys.NumOfMeas,1)/1000; % 5% gaussian noise

else
    CalNoise=zeros(sys.NumOfMeas,1);
end
end
% Plant Noise Generator
function [CalNoise]=CalStateNoise(currentstate)
global sys
    CalNoise = randn(sys.NumOfStates,1)/1000; % 5% gaussian noise
end
```

Model and SMC design

```
%working with two output, two input, can be extended easily
function [zetaTrans,fx,gx,hx,hmx,Crx,Drx,Ax,Cx,order,Frdrivex,Fldrivex,...
RightCurrent,LeftCurrent]=CalNormForm(SysUncertainty,isELO)
syms r vr theta w mcll mcrr real

% load system data
GR = 20; %gear ratio
Rw = 0.0508; %m, radius of wheel
Br = 3.2e-5;% motor linear resistance coefficient
Rm = 2.6374; %ohm, motor resistance
rm = 0.036;  %Nm/A, motor constant
Vb = 12;%V, motor battery voltage, assume ideal battery
B  = 0.3683; %m, wheel axis width
m  = 3.6; %kg, DaNI mass
g  = 9.81;%gravity constant
```

```
fr = 0.02;% 5*0.004;%rolling resistance coefficient
Jm = 6.7e-6; %moment of inertia of motor at the motor side
Jw = 0.5*0.05*Rw^2; %moment of inertia of wheel
meff = m + 2/Rw^2*(Jm*GR^2+Jw); %effecive mass
Jeff = 1/12*m*B^2; %this nees to be tuned carefully, needs further action
Rb = 0.106; %20*5.3e-3; % nominal impedance of battery, 20 times

if(SysUncertainty && ~isEL0)
    Jeff=Jeff*1.05;
    fr = fr*1.1;
    Jm = Jm*0.95;
end
%%define system
Fxrf = -GR^2/Rw^2*(rm^2/Rm+Br)*(vr+B/2*w);%for f
Fxrg = GR*Vb*rm/Rw/Rm; %for g
Frr =  fr*m*g/2*sign(vr+w*B/2); %fr*m*g/2*tanh((vr+w*B/2)/0.001);
Fxlf = -GR^2/Rw^2*(rm^2/Rm+Br)*(vr-B/2*w);%for f
Fxlg = GR*Vb*rm/Rw/Rm; %for g
Frl = fr*m*g/2*sign(vr-w*B/2); % fr*m*g/2*tanh((vr-w*B/2)/0.001);%

Frdrivex = matlabFunction(Fxrg*mcrr + Fxrf ,'vars',[mcrr,vr,w]);
Fldrivex = matlabFunction(Fxlg*mcll + Fxlf,'vars',[mcll,vr,w]);

RightCurrent = matlabFunction(1/Rm*(Vb*mcrr-rm*(B/2*w+vr)*GR/Rw),...
'vars',[mcrr,vr,w]);
LeftCurrent  = matlabFunction(1/Rm*(Vb*mcll-rm*(-B/2*w+vr)*GR/Rw),...
'vars',[mcll,vr,w]);
x = [r vr theta w]'; %states
% sytem equation is  xdot = f + g*u; u=[mcr, mcl]';
f = [vr;
    1/meff*(Fxrf-Frr+Fxlf-Frl);
    w;
    B/2/Jeff*(Fxrf-Frr-Fxlf+Frl)];

g = [0  0;
     1/meff*Fxrg    1/meff*Fxlg;
     0  0;
     B/2/Jeff*Fxrg   -B/2/Jeff*Fxlg];
```

```
fx=matlabFunction(f, 'vars', x');
gx=matlabFunction(g, 'vars', x');

% define output
h = [r, theta]';
hx = matlabFunction(h,'vars',x');

%calculate norm form
[zeta,Cr,Dr] = CalLie(f,h,g,x);
order = length(zeta);
zetaTrans=matlabFunction(zeta, 'vars', x');

% Ax and Cx
Ax = jacobian(f,x);
Ax = matlabFunction(Ax,  'vars', x');
% define measurement
hm = [r,theta]';
hmx = matlabFunction(hm,'vars',x');

Cx = jacobian(hm,x);
Cx = matlabFunction(Cx,'vars',x');

% Crx and Drx
Crx=matlabFunction(Cr, 'vars', x');
Drx=matlabFunction(Dr, 'vars', x');

%Controllability and Observability
Ctr = [liebracket(f,g(:,1),x,1),liebracket(f,g(:,2),x,1)];
Ctrrank = rank(Ctr);

lobs = [liederivative(f,hm(1),x,1),liederivative(f,hm(2),x,1)]';
Obs = jacobian(lobs,x);
Obsrank = rank(Obs);

end
```

Appendix C: Code for discrete traffic control

Generate FTS

```
clear all
clc
clear
disp('this is traffic_lowlevel')
modelcase = 'turn1_Dishigh'; % 'turn1_Dishigh', 'turn2_Dislow','turn2_Dishigh'

[Xcap, Cflow,Signal,para,Dis]=trafficdef(modelcase);

[pp,qq,rr,ss,tt,uu] = ndgrid(0:Xcap(1),0:Xcap(2),0:Xcap(3),0:Xcap(4),...
0:Xcap(5),0:Xcap(6));

States_Mat = uint8([pp(:),qq(:),rr(:),ss(:),tt(:),uu(:)]);
N = uint16(size(States_Mat,1));
State_labels = cell(1,N);
States = cell(1,N);
for i=1:N
    k=1;
    States{i} = strcat('x',strrep(num2str(States_Mat(i,:)),' ',''));
    if States_Mat(i,1) <=Xcap(1)-1 State_labels{i}{k} = 'l1_low'; k=k+1; end
    if States_Mat(i,2) <=Xcap(2)-2 State_labels{i}{k} = 'l2_low'; k=k+1; end
    if States_Mat(i,3) <=Xcap(3)-2 State_labels{i}{k} = 'l3_low'; k=k+1; end
    if States_Mat(i,6) <=Xcap(6)-1 State_labels{i}{k} = 'l6_low'; k=k+1; end
end

save('States_Info','States','State_labels');
```

```
clear States;
clear State_labels;
clear i k pp qq rr ss tt uu
% pause
TS = false(N,N,length(Signal));
yindex = uint16(zeros(1,N));
tic
for i=uint16(1:length(Signal))
    SignalTem = Signal{i};
    parfor j=uint16(1:N)
         y = uint8(trafficdyn(Xcap,Cflow,States_Mat(j,:),SignalTem,para,Dis));
         [~,yindex(j)] = ismember(y,States_Mat,'rows');
    end
      TS(sub2ind(size(TS),1:N,yindex,uint16(ones(1,N))*i)) = 1;
end
toc
clear States_Mat yindex y i j

TSs123=TS(:,:,1);

TSs163=TS(:,:,2);

TSs4523=TS(:,:,3);

TSs4563=TS(:,:,4);

switch modelcase
    case 'turn1_Dislow'
        save('FTS_turn1_Dislow', 'TSs123','TSs163','TSs4523','TSs4563');
    case 'turn2_Dislow'
        save('FTS_turn2_Dislow', 'TSs123','TSs163','TSs4523','TSs4563');
    case 'turn1_Dishigh'
        save('FTS_turn1_Dishigh','TSs123','TSs163','TSs4523','TSs4563');
    case 'turn2_Dishigh'
        save('FTS_turn2_Dishigh','TSs123','TSs163','TSs4523','TSs4563');
    otherwise
        disp('something wrong in GenFTS');
end
```

Discrete control synthesis

```
import numpy as np
import scipy.io
from scipy import sparse as sp
from tulip import spec, synth, transys
from tulip.transys import machines
import matplotlib.pyplot as plt

env_actions = [
    {
        'name': 'env_actions',
        'values': transys.MathSet({'turn1_Dislow', 'turn2_Dislow',...
 'turn1_Dishigh', 'turn2_Dishigh'})
    }
]
# traffic signaling
sys_actions = [
    {
        'name': 'sys_actions',
        'values': transys.MathSet({'s123', 's163', 's4523', 's4563'})
    }
]
sys = transys.FTS(env_actions, sys_actions)

States_Info = scipy.io.loadmat('../project_mycode_matlab/...
FTS_lowlevel/States_Info.mat')
interstr1 = States_Info['States']
states = [str(''.join(letter)) for letter_array in interstr1[0] ...
for letter in letter_array]
atopro = {'l1_low', 'l2_low', 'l3_low', 'l6_low'}
sys.atomic_propositions.add_from(atopro)  # li_low means cars on link i <= low_limit
interstr2 = States_Info['State_labels']
state_labels = [[str(''.join(letter)) for letter_array in letter_aarray[0] for ...
letter in letter_array]
               for letter_aarray in interstr2[0]]

for state, label in zip(states, state_labels):
    sys.states.add(state, ap=label)
```

```
sys.states.initial.add(states[len(states)-1])
print(len(sys.states))
print(len(state_labels))
print('States Loaded Successfully!')
# load fts for turn1_Dislow
fts_turn1_Dislow_mat = scipy.io.loadmat('../project_mycode_matlab/...
FTS_lowlevel/FTS_turn1_Dislow.mat')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dislow_mat['TSs123']), states,
                        sys_actions='s123', env_actions='turn1_Dislow')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dislow_mat['TSs163']), states,
                        sys_actions='s163', env_actions='turn1_Dislow')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dislow_mat['TSs4523']), states,
                        sys_actions='s4523', env_actions='turn1_Dislow')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dislow_mat['TSs4563']), states,
                        sys_actions='s4563', env_actions='turn1_Dislow')
print('FTS1 loaded successfully!')

# load fts for turn1_Dishigh
fts_turn1_Dishigh_mat = scipy.io.loadmat('../project_mycode_matlab/...
FTS_lowlevel/FTS_turn1_Dishigh.mat')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dishigh_mat['TSs123']), states,
                        sys_actions='s123', env_actions='turn1_Dishigh')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dishigh_mat['TSs163']), states,
                        sys_actions='s163', env_actions='turn1_Dishigh')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dishigh_mat['TSs4523']), states,
                        sys_actions='s4523', env_actions='turn1_Dishigh')
sys.transitions.add_adj(sp.lil_matrix(fts_turn1_Dishigh_mat['TSs4563']), states,
                        sys_actions='s4563', env_actions='turn1_Dishigh')
print('FTS2 loaded successfully!')

# load fts for turn2_Dislow
fts_turn2_Dislow_mat = scipy.io.loadmat('../project_mycode_matlab...
/FTS_lowlevel/FTS_turn2_Dislow.mat')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dislow_mat['TSs123']), states,
                        sys_actions='s123', env_actions='turn2_Dislow')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dislow_mat['TSs163']), states,
                        sys_actions='s163', env_actions='turn2_Dislow')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dislow_mat['TSs4523']), states,
                        sys_actions='s4523', env_actions='turn2_Dislow')
```

```
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dislow_mat['TSs4563']), states,
                        sys_actions='s4563', env_actions='turn2_Dislow')
print('FTS3 loaded successfully!')

# load fts for turn2_Dishigh
fts_turn2_Dishigh_mat = scipy.io.loadmat('../project_mycode_matlab/...
FTS_lowlevel/FTS_turn2_Dishigh.mat')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dishigh_mat['TSs123']),
                        states, sys_actions='s123', env_actions='turn2_Dishigh')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dishigh_mat['TSs163']),
                        states, sys_actions='s163', env_actions='turn2_Dishigh')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dishigh_mat['TSs4523']),
                        states, sys_actions='s4523', env_actions='turn2_Dishigh')
sys.transitions.add_adj(sp.lil_matrix(fts_turn2_Dishigh_mat['TSs4563']),
                        states, sys_actions='s4563', env_actions='turn2_Dishigh')

# environment assumptions
env_vars = set()    # infer the rest from env_actions
env_init = set()        # empty set
env_prog = {'env_actions = "turn1_Dishigh"', 'env_actions = "turn2_Dishigh"'}
env_safe = {'((env_actions = "turn1_Dishigh") || (env_actions = "turn2_Dishigh"))...
 -> X ((env_actions = "turn1_Dislow") \
             || (env_actions = "turn2_Dislow")) '}

# system specifications
sys_vars = set()         # infer the rest from TS
sys_init = set()
sys_prog = {'(sys_actions = "s4523")  || (sys_actions = "s4563")'}
sys_prog |= {'(sys_actions = "s163") || (sys_actions = "s4563")'}
sys_prog |= {'l1_low', 'l2_low', 'l3_low', 'l6_low'}
sys_safe = {'l1_low -> X (l1_low)', 'l2_low -> X (l2_low)', 'l3_low -> X (l3_low)',...
 'l6_low -> X (l6_low)'}

specs = spec.GRSpec(env_vars, sys_vars, env_init, sys_init,
                    env_safe, sys_safe, env_prog, sys_prog)

print('FTS4 loaded successfully! Ready for control synthesis')

ctrl = synth.synthesize('gr1c', specs, sys=sys, ignore_sys_init=False)
```

```
samplerun = machines.random_run(ctrl, N=40)

print('synthesized planner has: n = ' + str(len(ctrl.states)) +
      'states. m =' + str(len(ctrl.transitions)) + 'transitions.')
runstates = samplerun[1]['loc']

X = np.zeros(shape=(len(runstates), 6))
for i in range(len(runstates)):
    X[i] = map(int,  runstates[i][1:])

t = np.arange(1, len(runstates)+1, 1)
trafficplot = plt.plot(t, X[:, 0], t, X[:, 1], t, X[:, 2], ...
t, X[:, 3], t, X[:, 4], t, X[:, 5])
plt.ylabel('Number of Cars')
plt.legend(iter(trafficplot), ('x1', 'x2', 'x3', 'x4', 'x5', 'x6'))
plt.savefig('trafficplot')

print('save controller')
# print(ctrl)
ctrl.save('traffic_network_lowlevel.eps')
```

Bibliography

[1] T. Luettel, "Autonomous Ground Vehicles Concepts and a Path to the Future", Proceedings of the IEEE, Vol. 100, pp. 1831-1839, 2012.

[2] K. Dresner, "Autonomous intersection management", Doctor of Philosophy, The University of Texas at Austin, 2009.

[3] U.S. Department of Transportation, "National Motor Vehicle Crash Causation Survey: Report to Congress (Report No. DOT HS-811-059)", National Highway Traffic Safety Administration, Washington, D.C, 2008.

[4] R. B. Allsop, "SIGSET: A computer program for calculating traffic capacity of signal-controlled road junctions", Traffic Eng. Control, vol. 12, no. 2, pp. 5860, 1971.

[5] J. Little, "The Synchronization of Traffic Signals by Mixed-Integer Linear Programming", Operations Research, vol. 14, no. 4, pp. 568-594, 1966.

[6] P. B. Hunt, D. L. Robertson, and R. D. Bretherton, "The SCOOT on-line traffic signal optimization technique", Traffic Eng. Control, vol. 23, pp. 190192, 1982.

[7] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu and R. Murray, "TuLiP: a software toolbox for receding horizon temporal logic planning", International Conference on Hybrid Systems: Computation and Control (HSCC), p. 313, 2011.

[8] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli and Y. Saar, "Synthesis of Reactive(1) designs", Journal of Computer and System Sciences, vol. 78, no. 3, pp. 911-938, 2012.

[9] K. Park, R. Zheng and X. Liu, "Cyber-physical systems: Milestones and research challenges", Computer Communications, vol. 36, no. 1, pp. 1-7, 2012.

[10] A. Alakeel, "Using Fuzzy Logic Techniques for Assertion-Based Software Testing Metrics", The Scientific World Journal, vol. 2015, pp. 1-8, 2015.

[11] G. Idler, "Model based assertions : a system design tool for cyber-physical systems", Master of Science in Engineering, The University of Texas at Austin, 2015.

[12] P. Derler, E. Lee and A. Vincentelli, "Modeling CyberPhysical Systems", Proceedings of the IEEE , vol.100, no.1, pp.13,28, 2012.

[13] S. Coogan, E. Aydin Gol, M. Arcak and C. Belta, "Traffic Network Control from Temporal Logic Specifications", IEEE Transactions on Control of Network Systems, pp. 1-1, 2015.

[14] C. Baier and J. Katoen, Principles of model checking. Cambridge, Mass.: MIT Press, 2008.

[15] J. Wu, A. Abbas-Turki and A. El Moudni, "Cooperative driving: an ant colony system for autonomous intersection management", Applied Intelligence, vol. 37, no. 2, pp. 207-222, 2011.

[16] M. Vasirani and S. Ossowski, "Evaluating policies for reservation-based intersection control", Proceedings of the 14th Portuguese Conference on Artificial Intelligence, vol. 14, pp.39-50, 2009.

[17] S. Huang, A. Sadek and Y. Zhao, "Assessing the Mobility and Environmental Benefits of Reservation-Based Intelligent Intersections Using an Integrated Simulator", IEEE Trans. Intell. Transport. Syst., vol. 13, no. 3, pp. 1201-1214, 2012.

[18] J. Kenney, "Dedicated Short-Range Communications (DSRC) Standards in the United States", Proceedings of the IEEE, vol. 99, no. 7, pp. 1162-1182, 2011.

[19] R. Verma and D. Vecchio, "Semiautonomous Multivehicle Safety", IEEE Robotics & Automation Magazine, vol. 18, no. 3, pp. 44-54, 2011.

[20] C. Wuthishuwong, A. Traechtler and T. Bruns, "Safe trajectory planning for autonomous intersection management by using vehicle to infrastructure communication", EURASIP J Wirel Commun Netw, vol. 2015, no. 1, p. 33, 2015.

[21] K. Fitzpatrick, M. Wooldridge and J. Blaschke, "Urban Intersection Design Guidance", Texas Department of Transportation, Product 0-4365-P2, vol. 1, 2004.

[22] L. Banda, M. Mzyece and G. Noel, "IP Mobility Support: Solutions for Vehicular Networks", IEEE Veh. Technol. Mag., vol. 7, no. 4, pp. 77-87, 2012.

[23] Fernandez, B. (2015). Nonlinear Control Systems (ME 384Q).